

CPSC 330

Applied Machine Learning

Lecture 1: Course Introduction

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

```
In [2]: 1 import glob
2 import os
3 import re
4 import sys
5
6 import matplotlib.pyplot as plt
7 import numpy as np
8 import pandas as pd
9
10 sys.path.append("../code/.")
11 import graphviz
12 import IPython
13 from IPython.display import HTML, display
14 from plotting_functions import *
15 from sklearn.dummy import DummyClassifier
16 from sklearn.feature_extraction.text import CountVectorizer
17 from sklearn.linear_model import LinearRegression, LogisticRegression
18 from sklearn.model_selection import train_test_split
19 from sklearn.pipeline import Pipeline, make_pipeline
20 from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor,
21 from utils import *
22
23 plt.rcParams["font.size"] = 16
24 pd.set_option("display.max_colwidth", 200)
```

Learning outcomes

From this lecture, you will be able to:

- explain the motivation to study machine learning;
- explain supervised machine learning;
- navigate through the course material;

- be familiar with the policies and how the class is going to run;
- set up your computer for the course.

Characters in this course?

- CPSC 330 teaching staff (3 instructors, including me, course coordinator Michelle Pang, and the TAs)
- Eva (a fictitious enthusiastic student)
- And you all, of course 😊 !

Meet your CPSC 330 instructor



- I am Mathias Lécuyer.
- You can call me Mathias.
- I am an Assistant Professor in Computer Science. I started in 2021, and this is my first time teaching an undergraduate class.
- I did my Ph.D. in Machine Learning, Security & Privacy, and System at Columbia University, in New York. I do research at the intersection(s) of these three areas.
- Contact information
 - Email: mathias.lecuyer@ubc.ca (<mailto:mathias.lecuyer@ubc.ca>)
 - Office: ICCS 317

Meet the whole teaching team

For the first time, CPSC 330 is delivered in multiple sections! Which means you have access to even more instructors. They are:



Giulia Toti - Section 201



Amir Abdi - Section 203

Look up Giulia's and Amir's office hours if you would like to get in touch with them. You can also send them a private message in Piazza.

We also have a course coordinator, Michelle Pang. Michelle will be your point of contact for admin related questions, such as regrade requests or concessions for serious personal matters. You can reach her at cpsc330-admin@cs.ubc.ca (<mailto:cpsc330-admin@cs.ubc.ca>)



Meet CPSC 330 TAs

We have many amazing TAs involved in the course this semester! They are:

- Bhandari Mihir
- Chan Miranda
- Fratzscher Anne-Sophie
- Gholami Peyman
- Johnson Ken
- Mak Serina
- Reid Elizabeth
- Rostami Ario
- Shpilevskiy Frederick
- Wong Eric
- Zarei Mahsa

They are available for you to contact on Piazza and during your tutorial session. We encourage you to say hi!

Meet Eva (a fictitious persona)!



Eva is among one of you. She has some experience in Python programming. She knows machine learning as a buzz word. During her recent internship, she has developed some interest and curiosity in the field. She wants to learn what is it and how to use it. She is a curious person and usually has a lot of questions!

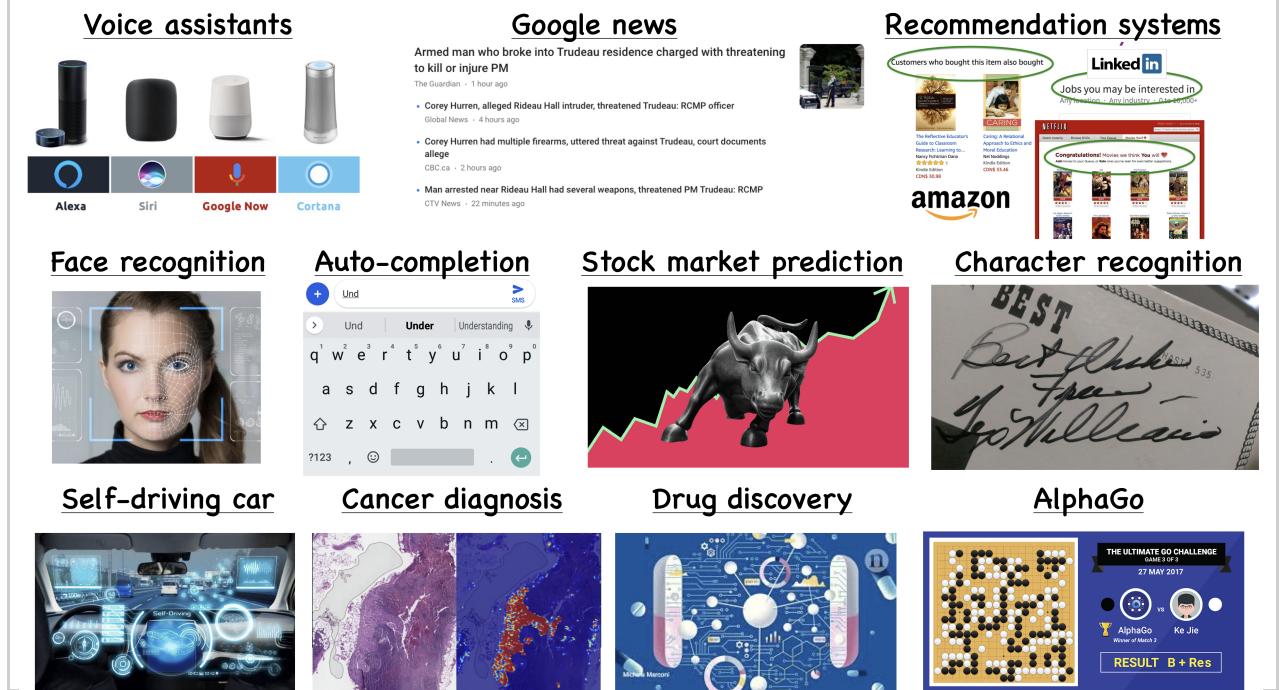
You all

- Since we're going to spend the semester with each other, I would like to know you a bit better.
- Let's try an interactive tool together to [share a bit about yourself](https://www.menti.com/aljv7bzq31z6) (<https://www.menti.com/aljv7bzq31z6>)

Why machine learning (ML)? [[video](https://www.youtube.com/watch?v=-1hTcS5ZE4w&t=1s) (<https://www.youtube.com/watch?v=-1hTcS5ZE4w&t=1s>)]

Prevalence of ML

Let's look at some examples.



- Image sources

- [Voice assistants](https://geeksfl.com/blog/best-voice-assistant/) (<https://geeksfl.com/blog/best-voice-assistant/>)
- [Google News](https://news.google.com) (<https://news.google.com>)
- [Recommendation systems](https://en.wikipedia.org/wiki/Recommender_system) (https://en.wikipedia.org/wiki/Recommender_system)
- [Face Recognition source](https://startupleague.online/blog/3dss-tech-facial-recognition-technology/) (<https://startupleague.online/blog/3dss-tech-facial-recognition-technology/>)
- [Auto-completion](https://9to5google.com/2020/08/10/android-11-autofill-keyboard/) (<https://9to5google.com/2020/08/10/android-11-autofill-keyboard/>)
- [Stock market prediction](https://hbr.org/2019/12/what-machine-learning-will-mean-for-asset-managers) (<https://hbr.org/2019/12/what-machine-learning-will-mean-for-asset-managers>)
- [Character recognition](https://en.wikipedia.org/wiki/Handwriting_recognition) (https://en.wikipedia.org/wiki/Handwriting_recognition)
- [AlphaGo](https://deepmind.com/alphago-china) (<https://deepmind.com/alphago-china>)
- [Self-driving cars](https://mc.ai/artificial-intelligence-in-self-driving-cars%E2%80%8A-%E2%80%8Ahow-far-have-we-gotten/) (<https://mc.ai/artificial-intelligence-in-self-driving-cars%E2%80%8A-%E2%80%8Ahow-far-have-we-gotten/>)
- [Drug discovery](https://www.nature.com/articles/d41586-018-05267-x) (<https://www.nature.com/articles/d41586-018-05267-x>)
- [Cancer detection](https://venturebeat.com/2018/10/12/google-ai-claims-99-accuracy-in-metastatic-breast-cancer-detection/) (<https://venturebeat.com/2018/10/12/google-ai-claims-99-accuracy-in-metastatic-breast-cancer-detection/>)

Saving time and scaling products

- Imagine writing a program for spam identification, i.e., whether an email is spam or non-spam.
- Traditional programming
 - Come up with rules using human understanding of spam messages.
 - Time consuming and hard to come up with robust set of rules.
- Machine learning
 - Collect large amount of data of spam and non-spam emails and let the machine learning algorithm figure out rules.
- With machine learning, you're likely to
 - Save time

Supervised machine learning

Types of machine learning

Here are some typical learning problems.

- **Supervised learning** ([Gmail spam filtering \(<https://support.google.com/a/answer/2368132?hl=en>\)](https://support.google.com/a/answer/2368132?hl=en)
 - Training a model from input data and its corresponding targets to predict targets for new examples.
- Unsupervised learning ([Google News \(<https://news.google.com/>\)](https://news.google.com/)
 - Training a model to find patterns in a dataset, typically an unlabeled dataset.
- Reinforcement learning ([AlphaGo \(<https://deepmind.com/research/case-studies/alphago-the-story-so-far>\)](https://deepmind.com/research/case-studies/alphago-the-story-so-far)
 - A family of algorithms for finding suitable actions to take in a given situation in order to maximize a reward.
- Recommendation systems ([Amazon item recommendation system \(<https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf>\)](https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf)
 - Predict the "rating" or "preference" a user would give to an item.

What is supervised machine learning (ML)?

- Training data comprises a set of observations (X) and their corresponding targets (y).
- We wish to find a model function f that relates X to y .
- We use the model function to predict targets of new examples.

Training data

Unseen test data

Example: Predict whether a message is spam or not**Input features X and target y**

Do not worry about the code and syntax for now.

Download SMS Spam Collection Dataset from [here \(<https://www.kaggle.com/uciml/sms-spam-collection-dataset>\)](https://www.kaggle.com/uciml/sms-spam-collection-dataset).

```
In [3]: 1 sms_df = pd.read_csv("../data/spam.csv", encoding="latin-1")
2 sms_df = sms_df.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
3 sms_df = sms_df.rename(columns={"v1": "target", "v2": "sms"})
4 train_df, test_df = train_test_split(sms_df, test_size=0.10, random_state=42)
5 HTML(train_df.head().to_html(index=False))
```

Out[3]:	target	sms
spam	LookAtMe! Thanks for your purchase of a video clip from LookAtMe!, you've been charged 35p. Think you can do better? Why not send a video in a MMSto 32323.	
ham	Aight, I'll hit you up when I get some cash	
ham	Don no da;)whats you plan?	
ham	Going to take your babe out ?	
ham	No need lar. Jus testing e phone card. Dunno network not gd i thk. Me waiting 4 my sis 2 finish bathing so i can bathe. Dun disturb u liao u cleaning ur room.	

Training a supervised machine learning model with X and y

```
In [4]: 1 X_train, y_train = train_df['sms'], train_df['target']
2 X_test, y_test = test_df['sms'], test_df['target']
3
4 clf = Pipeline(
5     [
6         ("vect", CountVectorizer(max_features=5000)),
7         ("clf", LogisticRegression(max_iter=5000)),
8     ]
9 )
10 clf.fit(X_train, y_train)
```

```
Out[4]: Pipeline(steps=[('vect', CountVectorizer(max_features=5000)),
 ('clf', LogisticRegression(max_iter=5000))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Predicting on unseen data using the trained model

```
In [5]: 1 pd.DataFrame(X_test.iloc[0:4])
```

	sms
3245	Funny fact Nobody teaches volcanoes 2 erupt, tsunamis 2 arise, hurricanes 2 sway aroundn no 1 teaches hw 2 choose a wife Natural disasters just happens
944	I sent my scores to sophas and i had to do secondary application for a few schools. I think if you are thinking of applying, do a research on cost also. Contact joke ogunrinde, her school is one m...
1044	We know someone who you know that fancies you. Call 09058097218 to find out who. POBox 6, LS15HB 150p
2484	Only if you promise your getting out as SOON as you can. And you'll text me in the morning to let me know you made it in ok.

Do not worry about the code and syntax for now.

In [6]:

```

1 pred_dict = {
2     "sms": X_test.iloc[0:4],
3     "spam_predictions": clf.predict(X_test.iloc[0:4]),
4 }
5 pred_df = pd.DataFrame(pred_dict)
6 pred_df.style.set_properties(**{"text-align": "left"})

```

Out[6]:

		sms	spam_predictions
3245	Funny fact Nobody teaches volcanoes 2 erupt, tsunamis 2 arise, hurricanes 2 sway aroundn no 1 teaches hw 2 choose a wife Natural disasters just happens		ham
944	I sent my scores to sophas and i had to do secondary application for a few schools. I think if you are thinking of applying, do a research on cost also. Contact joke ogunrinde, her school is one me the less expensive ones		ham
1044	We know someone who you know that fancies you. Call 09058097218 to find out who. POBox 6, LS15HB 150p		spam
2484	Only if you promise your getting out as SOON as you can. And you'll text me in the morning to let me know you made it in ok.		ham

We have accurately predicted labels for the unseen text messages above!

(Supervised) machine learning: popular definition

A field of study that gives computers the ability to learn without being explicitly programmed.
-- Arthur Samuel (1959)

ML is a different way to think about problem solving.

Logical and mathematical

Traditional programming

Example: Given a number n and a program to

Examples

Let's look at some concrete examples of supervised machine learning.

Do not worry about the code at this point. Just focus on the input and output in each example.

Example 1: Predicting whether a patient has a liver disease or not

Input data

Suppose we are interested in predicting whether a patient has the disease or not. We are given some tabular data with inputs and outputs of liver patients, as shown below. The data contains a number of input features and a special column called "Target" which is the output we are interested in predicting.

Download the data from [here](https://www.kaggle.com/uciml/indian-liver-patient-records) (<https://www.kaggle.com/uciml/indian-liver-patient-records>).

```
In [7]: 1 df = pd.read_csv("../data/indian_liver_patient.csv")
2 df = df.drop("Gender", axis=1)
3 df["Dataset"] = df["Dataset"].replace(1, "Disease")
4 df["Dataset"] = df["Dataset"].replace(2, "No Disease")
5 df.rename(columns={"Dataset": "Target"}, inplace=True)
6 train_df, test_df = train_test_split(df, test_size=4, random_state=42)
7 HTML(train_df.head().to_html(index=False))
```

Out[7]:	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Ami
	40	14.5	6.4	358		50
	33	0.7	0.2	256		21
	24	0.7	0.2	188		11
	60	0.7	0.2	171		31
	18	0.8	0.2	199		34

Building a supervise machine learning model

Let's train a supervised machine learning model with the input and output above.

In [8]:

```
1 from lightgbm.sklearn import LGBMClassifier
2 X_train = train_df.drop(columns=["Target"], axis=1)
3 y_train = train_df["Target"]
4 X_test = test_df.drop(columns=["Target"], axis=1)
5 y_test = test_df["Target"]
6 model = LGBMClassifier(random_state=123)
7 model.fit(X_train, y_train)
```

Out[8]: LGBMClassifier(random_state=123)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model predictions on unseen data

- Given features of new patients below we'll use this model to predict whether these patients have the liver disease or not.

In [9]:

```
1 HTML(X_test.reset_index(drop=True).to_html(index=False))
```

Out[9]:

Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Ami
19	1.4	0.8	178		13
12	1.0	0.2	719		157
60	5.7	2.8	214		412
42	0.5	0.1	162		155

In [10]:

```
1 pred_df = pd.DataFrame({"Predicted_target": model.predict(X_test).tolist()})
2 df_concat = pd.concat([pred_df, X_test.reset_index(drop=True)], axis=1)
3 HTML(df_concat.to_html(index=False))
```

Out[10]:

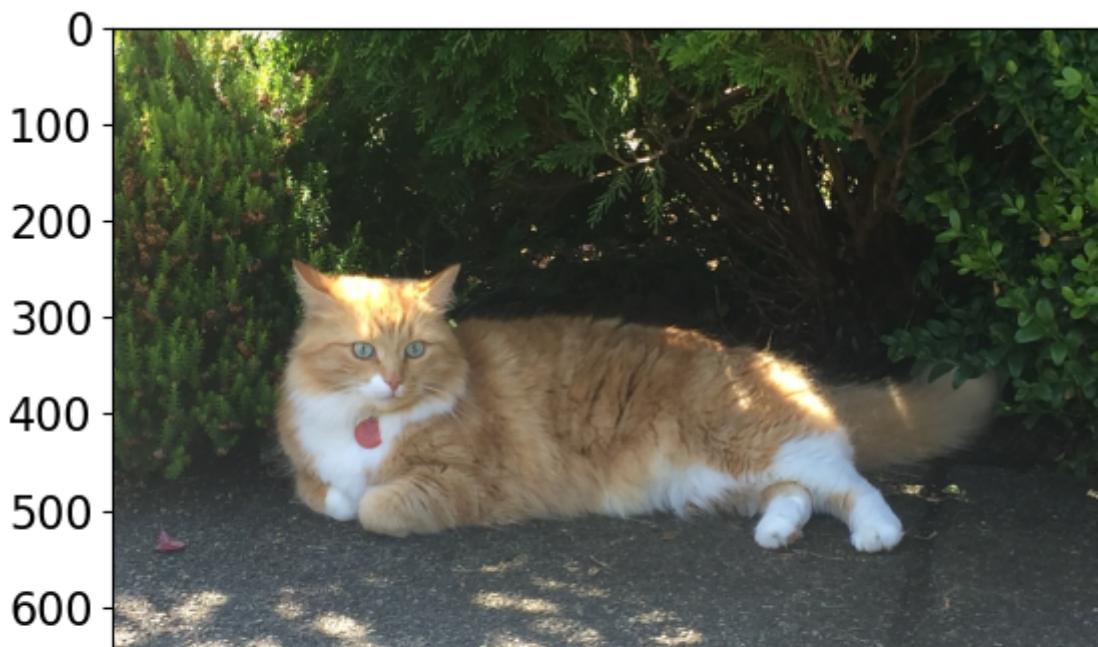
Predicted_target	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferas	
No Disease	19	1.4	0.8	178		1
Disease	12	1.0	0.2	719		15
Disease	60	5.7	2.8	214		41
Disease	42	0.5	0.1	162		15

Example 2: Predicting the label of a given image

Suppose you want to predict the label of a given image using supervised machine learning. We are using a pre-trained model here to predict labels of new unseen images.

In [11]:

```
1 from PIL import Image
2
3 # Predict labels with associated probabilities for unseen images
4 images = glob.glob("../data/test_images/*.*")
5 for image in images:
6     img = Image.open(image)
7     img.load()
8     plt.imshow(img)
9     plt.show()
10    df = classify_image_recent(img)
11    print(df.to_string(index=False))
12    print("-----")
```

**Example 3: Predicting sentiment expressed in a movie review**

Suppose you are interested in predicting whether a given movie review is positive or negative. You can do it using supervised machine learning.

Download the data from [here](https://www.kaggle.com/utathya/imdb-review-dataset) (<https://www.kaggle.com/utathya/imdb-review-dataset>).

In [12]:

```
1 imdb_df = pd.read_csv("../data/imdb_master.csv", encoding="ISO-8859-1")
2 imdb_df = imdb_df[imdb_df["label"].str.startswith(("pos", "neg"))]
3 imdb_df.drop(["Unnamed: 0", "type", "file"], axis=1, inplace=True)
4 imdb_df.rename(columns={"label": "target"}, inplace=True)
5 train_df, test_df = train_test_split(imdb_df, test_size=0.10, random_st
6 HTML(train_df.head().to_html(index=False))
```

Out[12]:

review target

It may have been inevitable that with the onslaught of "slasher" movies in the early 1980's, that a few good ones might slip through the cracks. This is a great "rare" film from Jeff Lieberman, who insured his cult status with his memorable 1970's films "Squirm" and "Blue Sunshine".
 Five young people head into the Oregon mountains (this movie was actually shot on location) to do some camping and check out the deed to some land that one of them has acquired. Before long, they will predictably be terrorized by a bulky killer with an incredibly creepy wheezing laugh.
 "Just Before Dawn" is noticeably more ambitious, "arty", and intelligent than some slasher films. Lieberman actually fleshes out the characters - well, two of them, anyway - as much as a 90-minute-long film will allow him. The film has genuine moments of suspense and tension, and actually refrains from graphic gore, save for one killing right at the beginning.
 There is an above-average cast here, including Oscar winner George Kennedy, as a forest ranger who's understandably gone a little flaky from having been alone in the wilderness for too long. Jack Lemmon's son Chris, future Brian De Palma regular Gregg Henry, blonde lead Deborah Benson (it's too bad she hasn't become a more well-known performer, judging by her work here), Ralph Seymour ("Ghoulies"), Mike Kellin ("Sleepaway Camp"), and Jamie Rose ("Chopper Chicks in Zombietown") round out the cast.
 Some of the shots are interesting, and the early music score by Brad Fiedel (now best known for his "Terminator" theme) is haunting and atmospheric.
 This is worth catching for the important plot twist at about the one hour mark, although a moment at about 75 minutes involving the heroine and a tree and the killer is almost comical; it may actually remind a viewer of a cartoon! One of the most clever touches is the final dispatching of the killer, which I'd never seen before in a horror film and probably won't see again.
 I didn't give it 10 out of 10 because I can't honestly say that I was that frightened. Still, it's an interesting slasher that is worthy of re-discovery.
 That deed don't mean nothing, son. Those mountains can't read.

9/10

pos

Tell the truth I'm a bit stunned to see all these positive reviews by so many people, which is also the main reason why I actually decided to see this movie. And after having seen it, I was really disappointed, and this comes from the guy that loves this genre of movie.
 I'm surprised at how all completely it is like a kid's movie with nudity for absolutely no reason and it all involves little children cursing and swearing. I'm not at all righteous but this has really gone too far in my account.
 Synopsis: The story about two guys sent to the big brother program for their reckless behavior. There they met up with one kid with a boob obsession and the other is a medieval freak.
 Just the name itself is not really connected with the story at all. They are not being a role model and/or doing anything but to serve their time for what they have done. The story is very predictable (though expected) and the humor is lame. And haven't we already seen the same characters (play by Mc Lovin') in so many other movies (like Sasquatch Gang?). I think I laughed thrice and almost fell asleep.
 Well the casting was alright after all he is the one that produced the screenplay. And the acting is so-so as expected when you're watching this type of movie. And the direction, what do one expect? This is the same guy who brought us Wet Hot American Summer, and that movie also sucks.
 But somehow he always managed to bring in some star to attract his horrendous movie.
 Anyway I felt not total riff off but a completely waste of time. Only the naked scenes seem to be the best part in the movie. Can't see any point why I should recommend this to anyone.
 Pros: Elizabeth Banks? Two topless scenes.
 Cons: Not funny, dreadful story, nudity and kids do not mix together.
 Rating: 3.5/10 (Grade: F)

neg

After getting thrown out of their last job and finding employment scarce in the United Kingdom, the six members of the Wonder Boys, better known as The Crazy Gang see an advertisement for employment in the gold strike town of Red Gulch in the Yukon Territory. It's from a newspaper clipping and on the back there's a story about Chamberlain saying the country better be prepared for war. Off they go to the Yukon and The Frozen Limits.
 By the way, it's a case of misplaced Chamberlains. The clipping is forty years old and it refers to Joe Chamberlain and the Boer War rather than Neville in the current crisis. But that's typical of how things go for this crew. I can see Stan Laurel making the same mistake.
 Of course when they get there it's a ghost town inhabited only by young Jean Kent and her grandfather Moore Marriott. He's getting on in years and is a bit touched in the head. Marriott's got a gold mine that he's misplaced somewhere that he goes to in his sleep, that is when he's sleepwalking.
 The Gang better help him find that mine or otherwise pretty Ms. Kent won't marry stalwart trapper Anthony Hulme, but rather saloon owner Bernard Lee, a fate worse than death.
 This was my first exposure to the Crazy Gang and I can see both why they were so acclaimed in the UK and why they never made any impact across the pond. The jokes come fast and furious and then were a number of things that the Code in the USA just wouldn't allow. The jokes are also strictly topical British and a lot just wouldn't be gotten over here.
 The sight gags are universal, the final chase scene is worthy of anything that the Marx Brothers did in America. My suggestion is that if you watch The Frozen Limits, tape it if you have a working familiarity with British history and run it two or three times just to make sure you pick up everything. It will be worth it.

pos

The plot for a movie such of this is a giveaway. How can you go wrong with a gay plot line and all the colors and music of India - a story like this writes itself. I'll watch most anything, but this was unwatchable. The sad thing is, the white folks are the most colorful in the film. Vanessa was a riot with a mouth like a sailor, and Jack was great eye candy, but everyone else was so boring. Saeed Jeffrey, who was exceptional in My Beautiful Landrette, did what he could but the story was so boring. The saving grace was really the background music, which made it OK to laugh at the film, instead of with the film, or not at all. There are many other better gay movies, ethnic movies, just plain movies. I give a lot of low budget movies a pass, but this shouldn't have been made, or should have been made by someone else.

neg

It is a damn good movie,with some surprising twists,a good cast and a great script. Only a couple of stupid bits,like the Rasta hit-man scene (This guy's a professional?) but that has been commented on already. The fact I had only heard one guy at work mention it before, and did not have many opinions or reviews to go on, made it even more entertaining. This gets a higher score than maybe some people think it deserves, but I have to factor in the low budget and the good effort from the cast. It sickens me that some movies get made whose budget equals the GDP of a small country,with a hyped up release,good reviews,an Oscar winning director and/or actors, and turn out to be so disappointing,with actors sleepwalking through their roles and uninspired directing,with predictable plot lines and a story with holes in it so big,Sandra Bullock could drive a bomb-loaded bus through it. (Examples in my opinion are The Terminal,Castaway,Matrix:Revolutions) Extra points are awarded for the wardrobe department choosing great clothes for the cast,especially Paulina Porizcova,who wears a rubber dress in one scene, and a jacket with "c*nt" on the back in large letters in another!A sex scene which shows off her tight ass and a good soundtrack are added bonuses! And PLEASE,enough with the Tarantino comparisons,this did not remind me of a Tarantino flick at all.... and Tarantino borrows virtually every idea he has ever had from other movies! Even if that is your opinion,are we saying once a certain film or book is written or directed one way,no-one can ever use the same ideas again? get real. This film has it's own style.

pos

In [13]:

```

1 # Build an ML model
2 X_train, y_train = train_df[ "review" ], train_df[ "target" ]
3 X_test, y_test = test_df[ "review" ], test_df[ "target" ]
4
5 clf = Pipeline(
6     [
7         ( "vect" , CountVectorizer(max_features=5000)),
8         ( "clf" , LogisticRegression(max_iter=5000)),
9     ]
10 )
11 clf.fit(X_train, y_train)

```

Out[13]: Pipeline(steps=[('vect', CountVectorizer(max_features=5000)), ('clf', LogisticRegression(max_iter=5000))])

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [14]:

```
1 # Predict on unseen data using the built model
2 pred_dict = {
3     "reviews": x_test.iloc[0:4],
4     "sentiment_predictions": clf.predict(x_test.iloc[0:4]),
5 }
6 pred_df = pd.DataFrame(pred_dict)
7 pred_df.style.set_properties(**{"text-align": "left"})
```

Out[14]:

reviews sentiment_predictions

	You'll feel like you've experienced a vacation in Hell after you have sat down and watched this horrible TV movie. This movie is an exercise in over-acting (very bad over-acting) to situations that made out to be more than what they are. I won't give away the plot, but once you realize why the people in this film are running from the native man in the film you will demand the two wasted hours of your life back. The only plus is seeing Marcia Brady running around in a bikini!	neg
11872	Bela Lugosi gets to play one of his rare good guy roles in a serial based upon the long running radio hit (which was also the source of a feature film where Lugosi played the villain.) Lugosi cuts a fine dashing figure and its sad that he didn't get more roles where he could be the guy in command in a good way. Here Chandu returns from the East in order to help the Princess Nadji who is being hunted by the leaders of the cult of Ubasti who need her to bring back from the dead the high priestess of their cult. This is a good looking globe trotting serial that is a great deal of fun. To be certain the pacing is a bit slack, more akin to one of Principals (the producing studios) features then a rip roaring adventure, but it's still enjoyable. This plays better than the two feature films that were cut from it because it allows for things to happen at their own pace instead of feeling rushed or having a sense that "hey I missed something". One of the trilogy of three good serials Lugosi made, the others being SOS Coast Guard and Phantom Creeps	pos
40828		

When you wish for the dragon to eat every cast member, you know you're in for a bad ride.

I went in with very, very low expectations, having read some of the other comments, and was not let down. Unlike some other cheap and failed movies, however, this one doesn't really remain hilariously (and unintentionally) funny throughout.

-SPOILERS FOLLOW-

First of all, plot it very inconsistent. Looking past the "small" mistakes, such as the dragon growing up in 3 hours, the whole idea it's based on is messed up. See, the movie wants us to believe that dragons came from outer space in the form of meteorites which really were dragon eggs. After explaining this, they show some peasant poking at one with his pitchfork and the dragon pops out.

Later, the obligatory "crazy scientist" guy babbles on about how dragons outlived the dinosaurs. So apparently humans were around when dinosaurs were, or we just have a fine little plot hole here. The other major thing is that the lab is blown up with a force "half as strong" as what was used for Hiroshima. Then two guys later walk in to check everything out, and it's almost unscathed! There's even another dragon, which grew out of who knows what. All in all it's very predictable. As soon as the guy mentioned cloning, I guessed they'd clone a dragon. That means that our Mr. Smarty-pants security guy isn't so intuitive and smart as the movie would have you believe, if you ignore that I knew this film would be about, you know, dragons.

36400

neg

Putting that aside, the second worst thing is the "special effects." Others have mentioned the fake rocks falling during the beginning, the CG helicopter, and the dragon. It looks a bit better than a blob, but it ruined whatever it had going for it when it trudged down the hall in the same manner time after time. To their credit, the flying dragons in the beginning looked OK from far away (although the one in the cave is probably the worst one in the whole movie.) These things are funny to watch, however. The scenes where a million different shots of the same person facing different ways are shown are not. Nor are the "introduction" screens with the vital stats.

Coming to the actors, they weren't the greatest, but I guess at least they tried? They seemed more enthusiastic about what they were doing than many of the actors participating in the recent "BloodRayne," for example, and you've got to give them points for that. One thing I noticed though was that the woman who plays Meredith often had her face covered in make-up that was many tones lighter than the rest of her. She looked like she had a bad run-in with some white-face.

The script is bad and cheesy. You don't really notice the music, but it's actually not too bad for the most part.

The bottom line is don't watch it unless you want to see it because you hear it's bad (like I did), although the only funny things are the bad CG effects. Other than that, don't waste your time and money.

Sorry, but Jacqueline Hyde (get it??? - Jack L and Hyde - Jekyll & Hyde) has some of the worst acting this side of hardcore porn, not to mention a script apparently written by a first-grader with undiagnosed learning disabilities.

Jackie Hyde inherits an old mansion by a grandfather she never knew she had. Guess who? Yes, an inventor of the special formula that slowly takes over one's body and mind - yes, that Mr. Hyde!

5166

neg

Despite some nice skin scenes, this film fails to register any feeling or emotion other than uncontrollable laughter.

As much as poor Jackie tries she just can't stay away from granddaddy's special formula and the result is an hour and half of wasted time.

Example 4: Predicting housing prices

Suppose we want to predict housing prices given a number of attributes associated with houses.

Download the data from [here \(<https://www.kaggle.com/harlfoxem/housesalesprediction>\)](https://www.kaggle.com/harlfoxem/housesalesprediction).

```
In [ ]: 1 df = pd.read_csv("../data/kc_house_data.csv")
2 df.drop(["id", "date"], axis=1, inplace=True)
3 df.rename(columns={"price": "target"}, inplace=True)
4 train_df, test_df = train_test_split(df, test_size=0.2, random_state=4)
5 HTML(train_df.head().to_html(index=False))
```

```
In [ ]: 1 # Build a regression model
2 import xgboost as xgb
3 from xgboost import XGBRegressor
4
5 X_train, y_train = train_df.drop("target", axis=1), train_df["target"]
6 X_test, y_test = test_df.drop("target", axis=1), train_df["target"]
7
8 model = XGBRegressor()
9 model.fit(X_train, y_train);
```

```
In [ ]: 1 # Predict on unseen examples using the built model
2 pred_df = pd.DataFrame(
3     {"Predicted target": model.predict(X_test[0:4]).tolist(), "Actual": X_test[0:4].reset_index(drop=True).values.tolist(),
4      "Predicted_target": model.predict(X_test[0:4]).tolist()})
5 )
6 df_concat = pd.concat([pred_df, X_test[0:4].reset_index(drop=True)], axis=1)
7 HTML(df_concat.to_html(index=False))
```

To summarize, supervised machine learning can be used on a variety of problems and different kinds of data.

[What would you like to use Machine Learning for? \(<https://www.menti.com/alk991nryvye>\)](https://www.menti.com/alk991nryvye)

🤔 Eva's questions

At this point, Eva has many questions.

- How are we exactly "learning" whether a message is spam?
- What do you mean by "learn without being explicitly programmed"? The code has to be somewhere ...
- Are we expected to get correct predictions for all possible messages? How does it predict the label for a message it has not seen before?
- What if the model mis-labels an unseen example? For instance, what if the model incorrectly predicts a non-spam as a spam? What would be the consequences?
- How do we measure the success or failure of spam identification?
- If you want to use this model in the wild, how do you know how reliable it is?
- Would it be useful to know how confident the model is about the predictions rather than just a yes or a no?

It's great to think about these questions right now. But Eva has to be patient. By the end of this course you'll know answers to many of these questions!



Machine learning workflow

Supervised machine learning is quite flexible; it can be used on a variety of problems and different kinds of data. Here is a typical workflow of a supervised machine learning systems.

What question do I want to answer?



Formulation to supervised machine learning problem

7. Test data predictions,
visualizations,
possible deployment



1. Data collection



2. Data cleaning, splitting



Break (5 min)



- We will try to take a 5-minute break half way through every class.

About this course

Course website

<https://github.com/UBC-CS/cpsc330-2022W2> (<https://github.com/UBC-CS/cpsc330-2022W2>) is the most important link. Please read everything on there!

CPSC 330 vs. 340

Read https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/330_vs_340.md (https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/330_vs_340.md) which explains the difference between the two courses.

TLDR:

- 340: how do ML models work?
- 330: how do I use ML models?
- CPSC 340 has many prerequisites.
- CPSC 340 goes deeper but has a more narrow scope.
- I think CPSC 330 will be more useful if you just plan to apply basic ML.

Registration, waitlist and prerequisites

Please go through [this document](https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/course_info.md#registration) (https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/course_info.md#registration) carefully before contacting us about these issues. Even then, I am very unlikely to be able to help with registration, waitlist or prerequisite issues.

- This year, specifically:
 - If you see available restricted seats and wondering whether they are going to be open for registration or not, these seats are reserved for data science minor students. This is a new program and due to its novelty, there are some registration delays. So we all need to be patient. This is not under my control and I'm unlikely to be of help with this.
 - One thing I can promise is that we won't just forget about these seats. The capacity of this class is ~90 students. If you are still on the waitlist, you'll get in the class depending upon how many data science minor students register and how many students drop the class.

Course format

- Lectures are T/Th at 3:30pm.
- Sometimes there will be videos to watch before or during the lecture time. (Check the main course page to see if you are expected to watch videos before the class.)

- Weekly tutorials will be **office hour format** run by the TAs and are **completely optional**.
- We'll have one midterm and one final.

Communications

- Our main forum for getting help will be [Piazza](https://piazza.com/class/lcg06c2ncl06el) (<https://piazza.com/class/lcg06c2ncl06el>).
- Other forms of communications (Canvas, email...) will likely go unresponded.
- Let's all take 2 minutes to register (through the link or through the tab on Canvas)
- You must have a @ubc.ca email associated with your account. Unrecognizable accounts (other emails) will be dropped without warning.

Grades

- The grading breakdown is [here](https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/grades.md) (<https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/grades.md>). This page also explains the policy on challenging grades.
- You have one week to raise a concern from the time that your grades were posted, by contacting the course coordinator.
- A score of at least 50% in the Final Exam is required to pass the course.

First deliverables

- First homework assignment is due **this coming Monday**, Jan 16th, at 11:59pm. The assignment is available on [GitHub](https://github.com/UBC-CS/cpsc330-2022W2/tree/main/hw/hw1) (<https://github.com/UBC-CS/cpsc330-2022W2/tree/main/hw/hw1>).
- You must do the first homework assignment on your own.
- The [Syllabus quiz](https://canvas.ubc.ca/courses/106375/quizzes/584868) (<https://canvas.ubc.ca/courses/106375/quizzes/584868>) is available on Canvas and is due **this coming Monday**, Jan 16th, at 11:59pm.

Please read this entire document about [asking for help](https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/asking_for_help.md) (https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/asking_for_help.md). **TLDR:** Be nice.

Lecture and homework format: Jupyter notebooks

- This document is a [Jupyter notebook](https://jupyter.org/) (<https://jupyter.org/>), with file extension `.ipynb`.
- Confusingly, "Jupyter notebook" is also the original application that opens `.ipynb` files - but has since been replaced by **Jupyter lab**.
 - Some things might not work with the Jupyter notebook application.
 - The course setup/install instructions include Jupyter lab.

- Jupyter notebooks contain a mix of code, code output, markdown-formatted text (including LaTeX equations), and more.
 - When you open a Jupyter notebook in one of these apps, the document is "live", meaning you can run the code.

In []: 1 1+1

In []: 1 x = [1, 2, 3]
2 x[0] = 9999
3 x

- By default, Jupyter prints out the result of the last line of code, so you don't need as many `print` statements.
- In addition to the "live" notebooks, Jupyter notebooks can be statically rendered in the web browser, e.g. [this \(\[https://github.com/UBC-CS/cpsc330/blob/master/lectures/01_intro.ipynb\]\(https://github.com/UBC-CS/cpsc330/blob/master/lectures/01_intro.ipynb\)\)](https://github.com/UBC-CS/cpsc330/blob/master/lectures/01_intro.ipynb).
 - This can be convenient for quick read-only access, without needing to launch the Jupyter notebook/lab application.
 - But you need to launch the app properly to interact with the notebooks.

Lecture style

- Lots of code snippets in Jupyter.
- There will be some [YouTube videos](https://www.youtube.com/channel/UC40oUwJPrUmhsYdURk8OjqA) (<https://www.youtube.com/channel/UC40oUwJPrUmhsYdURk8OjqA>) to watch before or during the lecture.
- We will also try to work on some questions and exercises together during the class.
- All materials will be posted on the course website. Lecture notes will be posted right before each class.
- Lectures from the previous semester are available on [this Jupyter book \(<https://ubc-cs.github.io/cpsc330/README.html>\)](https://ubc-cs.github.io/cpsc330/README.html) and on previous course repositories.
- I cannot promise anything will stay the same from last year to this year, so watch out for differences.

Setting up your computer for the course

Recommended browser

- I'll test the course notebooks and exams, which we'll be doing via [Canvas](https://canvas.ubc.ca/courses/106375) (<https://canvas.ubc.ca/courses/106375>), on the following two browsers: Chrome and Firefox. So I recommend that you use one of these browsers for the course.
- You can install Chrome [here](https://www.google.com/chrome/) (<https://www.google.com/chrome/>).
- You can install Firefox [here](https://www.mozilla.org/en-US/firefox/new/) (<https://www.mozilla.org/en-US/firefox/new/>).

Python requirements/resources

We will primarily use Python in this course.

Here is the basic Python knowledge you'll need for the course:

- Basic Python programming
- Numpy
- Pandas
- Basic matplotlib
- Sparse matrices

Some of you will already know Python, others won't. Homework 1 is all about Python.

We do not have time to teach all the Python we need but you can find some useful Python resources [here \(\[https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/python_notes.ipynb\]\(https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/python_notes.ipynb\)\)](https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/python_notes.ipynb).

Activity

In this course, we will primarily be using Python, git, GitHub, Canvas, Gradescope, and Piazza. Let's set up your computers for the course.

- Follow the setup instructions [here \(<https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/setup.md>\)](https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/setup.md) to create a course conda environment on your computer.

Checklist for you before next class

- Are you able to access course [Canvas \(<https://canvas.ubc.ca/courses/106375>\)](https://canvas.ubc.ca/courses/106375)?
- Are you able to access [Gradescope \(<https://www.gradescope.ca/courses/9483>\)](https://www.gradescope.ca/courses/9483)? (If not, refer to the [Gradescope Student Guide \(<https://lhub.ubc.ca/guides/gradescope-student-guide/>\)](https://lhub.ubc.ca/guides/gradescope-student-guide/).)
- Are you able to access [course Piazza \(<https://piazza.com/class/lcg06c2ncl06el>\)](https://piazza.com/class/lcg06c2ncl06el)?
- Did you follow the setup instructions [here \(<https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/setup.md>\)](https://github.com/UBC-CS/cpsc330-2022W2/blob/main/docs/setup.md) to create a course conda environment on your computer?
- Did you complete the [Syllabus quiz on Canvas \(<https://canvas.ubc.ca/courses/106375/quizzes/584868>\)](https://canvas.ubc.ca/courses/106375/quizzes/584868)? (Due date: Monday, Jan 16th at 11:59pm)
- Are you almost finished or at least started with homework 1 (Due: Monday, Jan 16th at 11:59pm)

Summary

- Machine learning is a different paradigm for problem solving.
- Very often it reduces the time you spend programming and helps customizing and scaling your products.
- In supervised learning we are given a set of observations (X) and their corresponding targets (y) and we wish to find a model function f that relates X to y .
- You should be ready with the technology stack on your laptop now. If you were not able to do it during lecture time or you ran into trouble, post on Piazza or attend one of the tutorials or office hours.
- Carefully read the course website. Make sure to complete the surveys.
- **The teaching team is here to help you learn the material and succeed in the course!**
- This semester can be particularly difficult because of the mixed online/in person format and other issues related to the ongoing pandemic. Please reach out if you are going through hard time. I'll try my best to be accommodative and empathetic.
- Let's have fun learning this material together!



In []:

1

CPSC 330

Applied Machine Learning

Lecture 2: Terminology, Baselines, Decision Trees

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

```
In [1]: 1 %load_ext autoreload
2 %autoreload 2
3
4 import glob
5 import os
6 import re
7 import sys
8 from collections import Counter, defaultdict
9
10 import matplotlib.pyplot as plt
11 import numpy as np
12 import pandas as pd
13
14 sys.path.append("../code/.")
15 import graphviz
16 import IPython
17 from IPython.display import HTML, display
18 from plotting_functions import *
19 from sklearn.dummy import DummyClassifier
20 from sklearn.feature_extraction.text import CountVectorizer
21 from sklearn.linear_model import LinearRegression, LogisticRegression
22 from sklearn.model_selection import train_test_split
23 from sklearn.pipeline import Pipeline, make_pipeline
24 from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor,
25 from utils import *
26
27 plt.rcParams["font.size"] = 16
28 pd.set_option("display.max_colwidth", 200)
```

Announcements

- Reminder of first deliverables:

- Syllabus quiz: Jan 16, 11:59pm
- Homework 1 (hw1): Jan 16, 11:59pm
- You can find the tentative due dates for all deliverables [here \(<https://github.com/UBC-CS/cpsc330-2022W2#lecture-schedule-tentative>\)](https://github.com/UBC-CS/cpsc330-2022W2#lecture-schedule-tentative).
- Please monitor [Piazza \(<https://piazza.com/class/lcg06c2ncl06el>\)](https://piazza.com/class/lcg06c2ncl06el) (especially pinned posts and instructor posts) for announcements.
- Let's check your status: <https://www.menti.com/alwcbyyjf26j> (<https://www.menti.com/alwcbyyjf26j>)

Quick recap: True or False?

(Same menti as before: <https://www.menti.com/alwcbyyjf26j> (<https://www.menti.com/alwcbyyjf26j>))

- There are different types of machine learning problems.
- Predicting spam and predicting housing prices are both examples of supervised machine learning.
- For problems such as spelling correction, translation, face recognition, spam identification, if you are a domain expert, it's usually faster and scalable to come up with a robust set of rules manually rather than building a machine learning model.
- Google News is likely using machine learning to organize news.

Learning outcomes

From this lecture, you will be able to

Terminology [[video \(<https://youtu.be/YNT8n4cXu4A>\)](https://youtu.be/YNT8n4cXu4A)]

- identify whether a given problem could be solved using supervised machine learning or not;
- differentiate between supervised and unsupervised machine learning;
- explain machine learning terminology such as features, targets, predictions, training, and error;
- differentiate between classification and regression problems;

Baselines [[video \(<https://youtu.be/6eT5cLL-2Vc>\)](https://youtu.be/6eT5cLL-2Vc)]

- use `DummyClassifier` and `DummyRegressor` as baselines for machine learning problems;
- explain the `fit` and `predict` paradigm and use `score` method of ML models;

Decision trees [[video \(<https://youtu.be/Hcf19Ij35rA>\)](https://youtu.be/Hcf19Ij35rA)]

- broadly describe how decision tree prediction works;
- use `DecisionTreeClassifier` and `DecisionTreeRegressor` to build decision trees using `scikit-learn`;

- visualize decision trees;

More terminology [[video](https://youtu.be/KEtsfXn4w2E) (<https://youtu.be/KEtsfXn4w2E>)]

- explain the difference between parameters and hyperparameters;
- explain the concept of decision boundaries;
- explain the relation between model complexity and decision boundaries.

Big picture and datasets

In this lecture, we'll talk about our first machine learning model: Decision trees. We will also familiarize ourselves with some common terminology in supervised machine learning.

Toy datasets

Later in the course we will use larger datasets from Kaggle, for instance. But for our first couple of lectures, we will be working with the following three toy datasets:

- [Quiz2 grade prediction classification dataset \(data/quiz2-grade-toy-classification.csv\)](#)
- [Quiz2 grade prediction regression dataset \(data/quiz2-grade-toy-regression.csv\)](#)
- [Canada USA cities dataset \(data/canada_usa_cities.csv\)](#)

If it's not necessary for you to understand the code, it will be in one of the files under the `code` directory to avoid clutter in this notebook. For example, most of the plotting code is going to be in `code/plotting_functions.py`.

Terminology [[video](https://youtu.be/YNT8n4cXu4A) (<https://youtu.be/YNT8n4cXu4A>)]

You will see a lot of variable terminology in machine learning and statistics. Let's familiarize ourselves with some of the basic terminology used in ML.

Check out [the accompanying video \(<https://youtu.be/YNT8n4cXu4A>\)](https://youtu.be/YNT8n4cXu4A) on this material.

We'll be using the following grade prediction toy dataset to demonstrate the terminology. Imagine that you are taking a course with four homework assignments and two quizzes. You and your friends are quite nervous about your quiz2 grades and you want to know how will you do based on your previous performance and some other attributes. So you decide to collect some data from your friends from last year and train a supervised machine learning model for quiz2 grade prediction.

In [2]:

```
1 classification_df = pd.read_csv("../data/quiz2-grade-toy-classification")
2 print(classification_df.shape)
3 classification_df.head()
```

(21, 8)

Out[2]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2	
0	1		1	92	93	84	91	92	A+
1	1		0	94	90	80	83	91	not A+
2	0		0	78	85	83	80	80	not A+
3	0		1	91	94	92	91	89	A+
4	0		1	77	83	90	92	85	A+

Recap: Supervised machine learning

Training data

X	y
😺	CAT
😺	CAT
...	...
🐶	DOG
🐕	DOG

Learning algorithm

Classification algorithm

ML model

Learned function f

Unseen test data

X	y
😺	?
🐕	?

Predictions
\hat{y}
CAT
DOG

Tabular data

In supervised machine learning, the input data is typically organized in a **tabular** format, where rows are **examples** and columns are **features**. One of the columns is typically the **target**.

Examples (n)	Features (d)							y
	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
1	1	91	93	88	92	94	A+	
1	0	78	87	88	85	80	not A+	
...
0	1	69	75	65	80	65	not A+	

Features : Features are relevant characteristics of the problem, usually suggested by experts. Features are typically denoted by X and the number of features is usually denoted by d .

Target : Target is the feature we want to predict (typically denoted by y).

Example : A row of feature values. When people refer to an example, it may or may not include the target corresponding to the feature values, depending upon the context. The number of examples is usually denoted by n .

Training : The process of learning the mapping between the features (X) and the target (y).

Example: Tabular data for grade prediction

The tabular data usually contains both: the features (x) and the target (y).

```
In [3]: 1 classification_df = pd.read_csv("../data/quiz2-grade-toy-classification"
2 classification_df.head()
```

Out[3]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	91	92	93	84	91	92	A+
1	1	0	94	90	80	83	91	not A+
2	0	0	78	85	83	80	80	not A+
3	0	1	91	94	92	91	89	A+
4	0	1	77	83	90	92	85	A+

So the first step in training a supervised machine learning model is separating x and y .

```
In [4]: 1 X = classification_df.drop(columns=["quiz2"])
2 y = classification_df["quiz2"]
3 X.head()
```

Out[4]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	
0	1		1	92	93	84	91	92
1	1		0	94	90	80	83	91
2	0		0	78	85	83	80	80
3	0		1	91	94	92	91	89
4	0		1	77	83	90	92	85

```
In [5]: 1 y.head()
```

Out[5]:

0	A+
1	not A+
2	not A+
3	A+
4	A+

Name: quiz2, dtype: object

```
In [6]: 1 X.shape
```

Out[6]: (21, 7)

Example: Tabular data for the housing price prediction

Here is an example of tabular data for housing price prediction. You can download the data from [here \(https://www.kaggle.com/harlfoxem/housesalesprediction\)](https://www.kaggle.com/harlfoxem/housesalesprediction).

```
In [7]: 1 housing_df = pd.read_csv("../data/kc_house_data.csv")
2 housing_df.drop(["id", "date"], axis=1, inplace=True)
3 HTML(housing_df.head().to_html(index=False))
```

Out[7]:

price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft
221900.0	3	1.00	1180	5650	1.0	0	0	3	7	
538000.0	3	2.25	2570	7242	2.0	0	0	3	7	
180000.0	2	1.00	770	10000	1.0	0	0	3	6	
604000.0	4	3.00	1960	5000	1.0	0	0	5	7	
510000.0	3	2.00	1680	8080	1.0	0	0	3	8	

```
In [8]: 1 X = housing_df.drop(columns=["price"])
2 y = housing_df["price"]
3 X.head()
```

Out[8]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above
0	3	1.00	1180	5650	1.0	0	0	3	7	1180
1	3	2.25	2570	7242	2.0	0	0	3	7	2170
2	2	1.00	770	10000	1.0	0	0	3	6	770
3	4	3.00	1960	5000	1.0	0	0	5	7	1050
4	3	2.00	1680	8080	1.0	0	0	3	8	1680

```
In [9]: 1 y.head()
```

Out[9]:

0	221900.0
1	538000.0
2	180000.0
3	604000.0
4	510000.0

Name: price, dtype: float64

To a machine, column names (features) have no meaning. Only feature values and how they vary across examples mean something.

Alternative terminology for examples, features, targets, and training

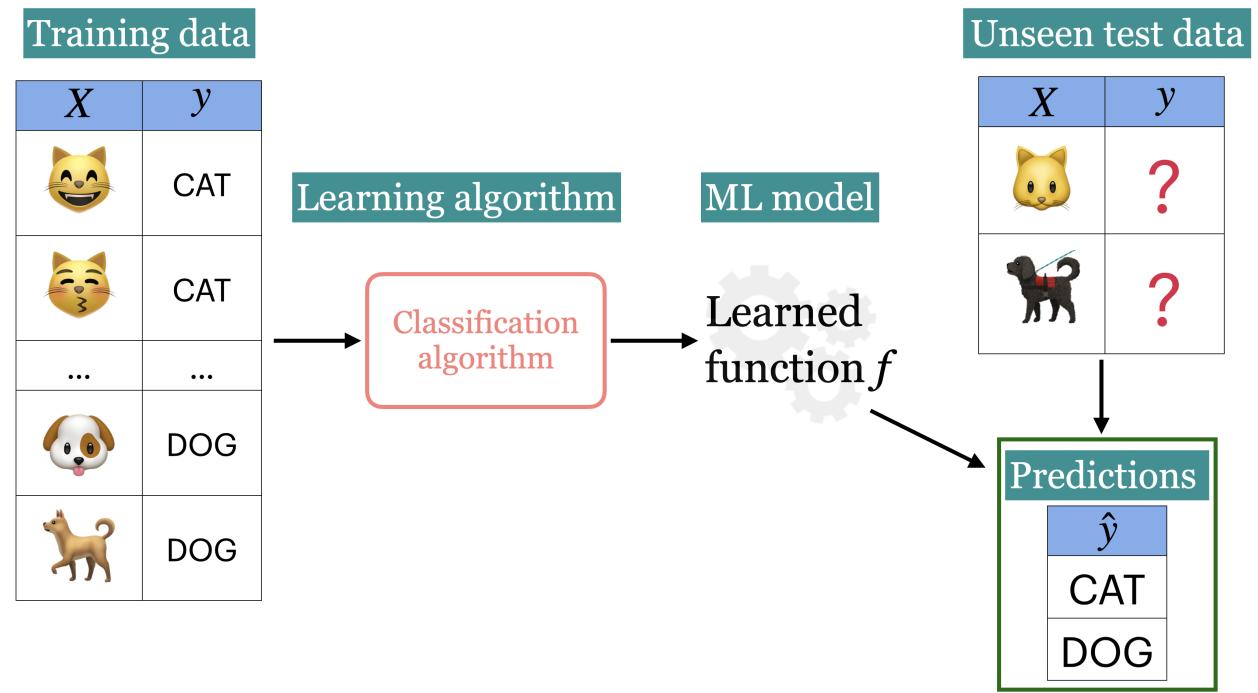
- **examples** = rows = samples = records = instances
- **features** = inputs = predictors = explanatory variables = regressors = independent variables = covariates
- **targets** = outputs = outcomes = response variable = dependent variable = labels (if categorical).
- **training** = learning = fitting

Check out [the MDS terminology document \(\[https://ubc-mds.github.io/resources_pages/terminology/\]\(https://ubc-mds.github.io/resources_pages/terminology/\)\)](https://ubc-mds.github.io/resources_pages/terminology/).

Let's pause our grades investigation for a second and think about the content of video [2.1](#) (<https://youtu.be/YNT8n4cXu4A>). In this video, you learned about the differences between unsupervised and supervised learning, and between classification and regression. The following slides are a quick recap.

Supervised learning vs. Unsupervised learning

In **supervised learning**, training data comprises a set of features (X) and their corresponding targets (y). We wish to find a **model function** f that relates X to y . Then use that model function **to predict the targets** of new examples.



In **unsupervised learning** training data consists of observations (X) **without any corresponding targets**. Unsupervised learning could be used to **group similar things together** in X or to provide **concise summary** of the data. We'll learn more about this topic in later videos.

Training data



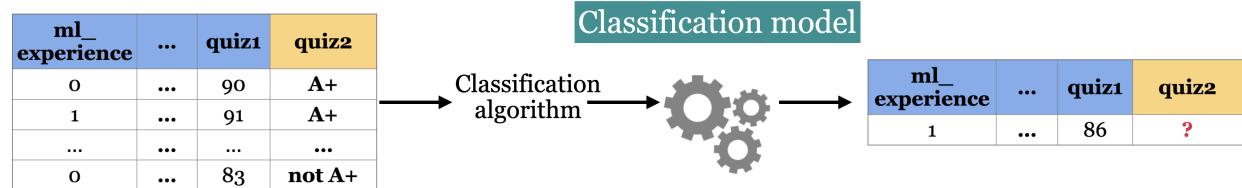
Supervised machine learning is about function approximation, i.e., finding the mapping function between x and y whereas unsupervised machine learning is about concisely describing the data.

Classification vs. Regression

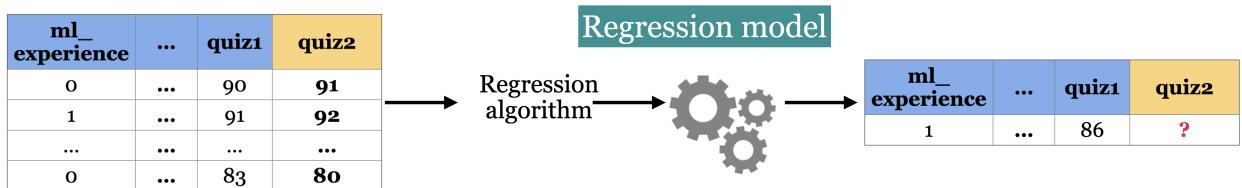
In supervised machine learning, there are two main kinds of learning problems based on what they are trying to predict.

- **Classification problem:** predicting among two or more discrete classes
 - Example1: Predict whether a patient has a liver disease or not
 - Example2: Predict whether a student would get an A+ or not in quiz2.
- **Regression problem:** predicting a continuous value
 - Example1: Predict housing prices
 - Example2: Predict a student's score in quiz2.

Predict whether a student would get an A+ or not in quiz2



Predict a student's score in quiz2



In [10]:

```

1 # quiz2 classification toy data
2 classification_df = pd.read_csv("../data/quiz2-grade-toy-classification"
3 classification_df.head(4)

```

Out[10]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2	
0	1		1	92	93	84	91	92	A+
1	1		0	94	90	80	83	91	not A+
2	0		0	78	85	83	80	80	not A+
3	0		1	91	94	92	91	89	A+

In [11]:

```

1 # quiz2 regression toy data
2 regression_df = pd.read_csv("../data/quiz2-grade-toy-regression.csv")
3 regression_df.head(4)

```

Out[11]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2	
0	1		1	92	93	84	91	92	90
1	1		0	94	90	80	83	91	84
2	0		0	78	85	83	80	80	82
3	0		1	91	94	92	91	89	92

?? Questions for you

Which of these are examples of supervised learning?

1. Finding groups of similar properties in a real estate data set.
2. Predicting real estate prices based on house features like number of rooms, learning from past sales as examples.
3. Grouping articles on different topics from different news sources (something like the Google News app).
4. Detecting credit card fraud based on examples of fraudulent and non-fraudulent transactions.

Let's answer here: <https://www.menti.com/alf1tszig9pa> (<https://www.menti.com/alf1tszig9pa>)

Which of these are examples of classification and which ones are of regression?

1. Predicting the price of a house based on features such as number of bedrooms and the year built.
2. Predicting if a house will sell or not based on features like the price of the house, number of rooms, etc.
3. Predicting percentage grade in CPSC 330 based on past grades.
4. Predicting whether you should bicycle tomorrow or not based on the weather forecast.

Let's answer here: <https://www.menti.com/alf1tszig9pa> (<https://www.menti.com/alf1tszig9pa>)

Baselines [[video](https://youtu.be/6eT5cLL-2Vc) (<https://youtu.be/6eT5cLL-2Vc>)]

Check out [the accompanying video](https://youtu.be/6eT5cLL-2Vc) (<https://youtu.be/6eT5cLL-2Vc>) on this material.

Supervised learning (Reminder)

- Training data → Machine learning algorithm → ML model
- Unseen test data + ML model → predictions

Training data

X	y
-----	-----

Unseen test data

X	y
-----	-----

Let's build a very simple supervised machine learning model for quiz2 grade prediction problem.

```
In [12]: 1 classification_df = pd.read_csv("../data/quiz2-grade-toy-classification")
2 classification_df.head()
```

Out[12]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2	
0	1		1	92	93	84	91	92	A+
1	1		0	94	90	80	83	91	not A+
2	0		0	78	85	83	80	80	not A+
3	0		1	91	94	92	91	89	A+
4	0		1	77	83	90	92	85	A+

```
In [13]: 1 classification_df['quiz2'].value_counts()
```

Out[13]:

not A+	11
A+	10
Name:	quiz2, dtype: int64

Seems like "not A+" occurs more frequently than "A+". What if we predict "not A+" all the time?

Baselines

Baseline : A simple machine learning algorithm based on simple rules of thumb.

- For example, most frequent baseline always predicts the most frequent label in the training set.
- Baselines provide a way to sanity check your machine learning model.

DummyClassifier

- sklearn's baseline model for classification
- Let's train `DummyClassifier` on the grade prediction dataset.

Steps to train a classifier using `sklearn`

1. Read the data
2. Create X and y
3. Create a classifier object
4. `fit` the classifier
5. `predict` on new examples
6. `score` the model

Reading the data

In [14]: 1 classification_df.head()

Out[14]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2	
0	1		1	92	93	84	91	92	A+
1	1		0	94	90	80	83	91	not A+
2	0		0	78	85	83	80	80	not A+
3	0		1	91	94	92	91	89	A+
4	0		1	77	83	90	92	85	A+

Create X and y

- $X \rightarrow$ Feature vectors
- $y \rightarrow$ Target

In [15]: 1 X = classification_df.drop(columns=["quiz2"])
2 y = classification_df["quiz2"]

Create a classifier object

- import the appropriate classifier
- Create an object of the classifier

In [16]: 1 from sklearn.dummy import DummyClassifier # import the classifier
2
3 dummy_clf = DummyClassifier(strategy="most_frequent") # Create a classifier

fit the classifier

- The "learning" is carried out when we call `fit` on the classifier object.

In [17]: 1 dummy_clf.fit(X, y); # fit the classifier

predict the target of given examples

- We can predict the target of examples by calling `predict` on the classifier object.

```
In [18]: 1 dummy_clf.predict(X) # predict using the trained classifier
```

```
Out[18]: array(['not A+', 'not A+', 'not A+', 'not A+', 'not A+', 'not A+',  
   'not A+', 'not A+', 'not A+', 'not A+', 'not A+', 'not A+',  
   'not A+', 'not A+', 'not A+', 'not A+', 'not A+', 'not A+',  
   'not A+', 'not A+', 'not A+'], dtype='|<U6')
```

score your model

- How do you know how well your model is doing?
- For classification problems, by default, `score` gives the **accuracy** of the model, i.e., proportion of correctly predicted targets.

$$\text{accuracy} = \frac{\text{correct predictions}}{\text{total examples}}$$

```
In [19]: 1 print("The accuracy of the model on the training data: %0.3f" % (dummy_
```

The accuracy of the model on the training data: 0.524

- Sometimes you will also see people reporting **error**, which is usually $1 - \text{accuracy}$
- `score`
 - calls `predict` on `X`
 - compares predictions with `y` (true targets)
 - returns the accuracy in case of classification.

```
In [20]: 1 print(  
2     "The error of the model on the training data: %0.3f" % (1 - dummy_c  
3 )
```

The error of the model on the training data: 0.476

fit, predict ,and score summary

Here is the general pattern when we build ML models using `sklearn`.

```
In [21]: 1 # Create `X` and `y` from the given data  
2 X = classification_df.drop(columns=["quiz2"])  
3 y = classification_df["quiz2"]  
4  
5 clf = DummyClassifier(strategy="most_frequent") # Create a class object  
6 clf.fit(X, y) # Train/fit the model  
7 print(clf.score(X, y)) # Assess the model  
8  
9 new_examples = [[0, 1, 92, 90, 95, 93, 92], [1, 1, 92, 93, 94, 92]]  
10 clf.predict(new_examples) # Predict on some new data using the trained
```

0.5238095238095238

```
Out[21]: array(['not A+', 'not A+'], dtype='|<U6')
```

DummyRegressor (<https://scikit-learn.org/0.15/modules/generated/sklearn.dummy.DummyRegressor.html>)

You can also do the same thing for regression problems using `DummyRegressor`.

If the DummyClassifier picked the most "popular" target, what should the DummyRegressor Pick?

- Let's build a regression baseline model using `sklearn`.

In [22]:

```
1 from sklearn.dummy import DummyRegressor
2
3 regression_df = pd.read_csv("../data/quiz2-grade-toy-regression.csv") #
4 X = regression_df.drop(columns=["quiz2"]) # Create `X` and `y` from the
5 y = regression_df["quiz2"]
6 reg = DummyRegressor() # Create a class object; uses mean as default st
7 reg.fit(X, y) # Train/fit the model
8 print(reg.score(X, y)) # Assess the model
9 new_examples = [[0, 1, 92, 90, 95, 93, 92], [1, 1, 92, 93, 94, 92]]
10 reg.predict(new_examples) # Predict on some new data using the trained
```

0.0

Out[22]: `array([86.28571429, 86.28571429])`

- The fit and predict paradigms similar to classification. The `score` method in the context of regression returns somethings called R^2 [score \(https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score\)](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score). (More on this in later videos.)
 - The maximum R^2 is 1 for perfect predictions.
 - For `DummyRegressor` it returns the mean of the `y` values.

In [23]:

```
1 reg.score(X, y)
```

Out[23]: 0.0

?? Questions for you

- Order the steps below to build ML models using `sklearn`.
 - `score` to evaluate the performance of a given model
 - `predict` on new examples
 - Creating a model instance
 - Creating `X` and `y`
 - `fit`
- Why do we train a Dummy Classifier?

Let's answer here: <https://www.menti.com/alvzb1o23znq>

Decision trees [[video \(<https://youtu.be/Hcf19lj35rA>\)](https://youtu.be/Hcf19lj35rA)]

Check out [the accompanying video \(<https://youtu.be/Hcf19lj35rA>\)](https://youtu.be/Hcf19lj35rA) on this material.

Writing a traditional program to predict quiz2 grade

- Can we do better than the baseline?
- Forget about ML for a second. If you are asked to write a program to predict whether a student gets an A+ or not in quiz2, how would you go for it?
- For simplicity, let's binarize the feature values.

$1 = \text{"yes"}; 0 = \text{"no"} \quad 1 = \text{"A+"; } 0 = \text{"not A+"}$

ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1	1	1	0	1	1	A+
0	0	1	1	0	0	1	not A+
1	0	0	1	0	1	0	not A+
0	1	0	0	1	1	0	A+
1	0	0	1	1	1	1	A+
0	1	0	1	0	0	0	not A+

- Is there a pattern that distinguishes yes's from no's and what does the pattern say about today?
- How about a rule-based algorithm with a number of *if* *else* statements?

```
if class_attendance == 1 and quiz1 == 1:
    quiz2 == "A+"
elif class_attendance == 1 and lab3 == 1 and lab4 == 1:
    quiz2 == "A+"
...

```

- How many possible rule combinations there could be with the given 7 binary features?
 - Gets unwieldy pretty quickly

Decision tree algorithm

- A machine learning algorithm to derive such rules from data in a principled way.
- Have you ever played [20-questions game \(\[https://en.wikipedia.org/wiki/Twenty_questions\]\(https://en.wikipedia.org/wiki/Twenty_questions\)\)?](https://en.wikipedia.org/wiki/Twenty_questions)
Decision trees are based on the same idea!
- Let's fit a decision tree using scikit-learn and predict with it.
- Recall that scikit-learn uses the term fit for training or learning and uses predict for prediction.

Building decision trees with sklearn

Let's binarize our toy dataset for simplicity.

```
In [24]: 1 classification_df = pd.read_csv("../data/quiz2-grade-toy-classification"
2 X = classification_df.drop(columns=["quiz2"])
3 y = classification_df["quiz2"]
4
5 X_binary = X.copy()
6 columns = ["lab1", "lab2", "lab3", "lab4", "quiz1"]
7 for col in columns:
8     X_binary[col] = X_binary[col].apply(lambda x: 1 if x >= 90 else 0)
9 X_binary.head()
```

```
Out[24]:   ml_experience  class_attendance  lab1  lab2  lab3  lab4  quiz1
          0             1                  1    1    1    0    1    1
          1             1                  0    1    1    0    0    1
          2             0                  0    0    0    0    0    0
          3             0                  1    1    1    1    1    0
          4             0                  1    0    0    1    1    0
```

```
In [25]: 1 y.head()
```

```
Out[25]: 0      A+
1  not A+
2  not A+
3      A+
4      A+
Name: quiz2, dtype: object
```

DummyClassifier on quiz2 grade prediction toy dataset

```
In [26]: 1 dummy_clf = DummyClassifier(strategy="most_frequent")
2 dummy_clf.fit(X_binary, y)
3 dummy_clf.score(X_binary, y)
```

Out[26]: 0.5238095238095238

DecisionTreeClassifier on quiz2 grade prediction toy dataset

```
In [27]: 1 from sklearn.tree import DecisionTreeClassifier
2
3 model = DecisionTreeClassifier() # Create a decision tree
4 model.fit(X_binary, y) # Fit a decision tree
5 model.score(X_binary, y) # Assess the model
```

Out[27]: 0.9047619047619048

The decision tree classifier is giving much higher accuracy than the dummy classifier. That's good news!

```
In [28]: 1 display_tree(X_binary.columns, model, counts=True) # model visualizatio
```

Out[28]: <graphviz.sources.Source at 0x104c1fdc0>

```
In [30]: 1 classification_df[
2     (classification_df["lab4"]>=90)
3     & (classification_df["class_attendance"] > .5)
4     & (classification_df["lab2"]>=90)
5     & (classification_df["quiz1"]<90)
6     # & (classification_df["ml_experience"]> .5)
7 ]
```

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2	
3	0		1	91	94	92	91	89	A+
9	1		1	95	95	94	94	85	not A+
14	0		1	95	90	93	95	70	not A+
20	1		1	96	92	92	96	87	A+

Some terminology related to trees

Here is a commonly used terminology in a typical representation of decision trees.

A root node : represents the first condition to check or question to ask

A branch : connects a node (condition) to the next node (condition) in the tree. Each branch typically represents either true or false.

An internal node : represents conditions within the tree

A leaf node : represents the predicted class/value when the path from root to the leaf node is followed.

How does predict work?

```
In [28]: 1 new_example = np.array([[0, 1, 0, 0, 1, 1, 1]])
2 new_example = pd.DataFrame(data=new_example, columns=x.columns)
3 new_example
```

```
Out[28]: ml_experience  class_attendance  lab1  lab2  lab3  lab4  quiz1
          0                 0           1     0     0     1     1     1
```

```
In [52]: 1 display_tree(X_binary.columns, model)
```

```
Out[52]: <graphviz.sources.Source at 0x1870424a0>
```

What's the prediction for the new example?

```
In [30]: 1 model.predict(new_example)
```

```
Out[30]: array(['A+'], dtype=object)
```

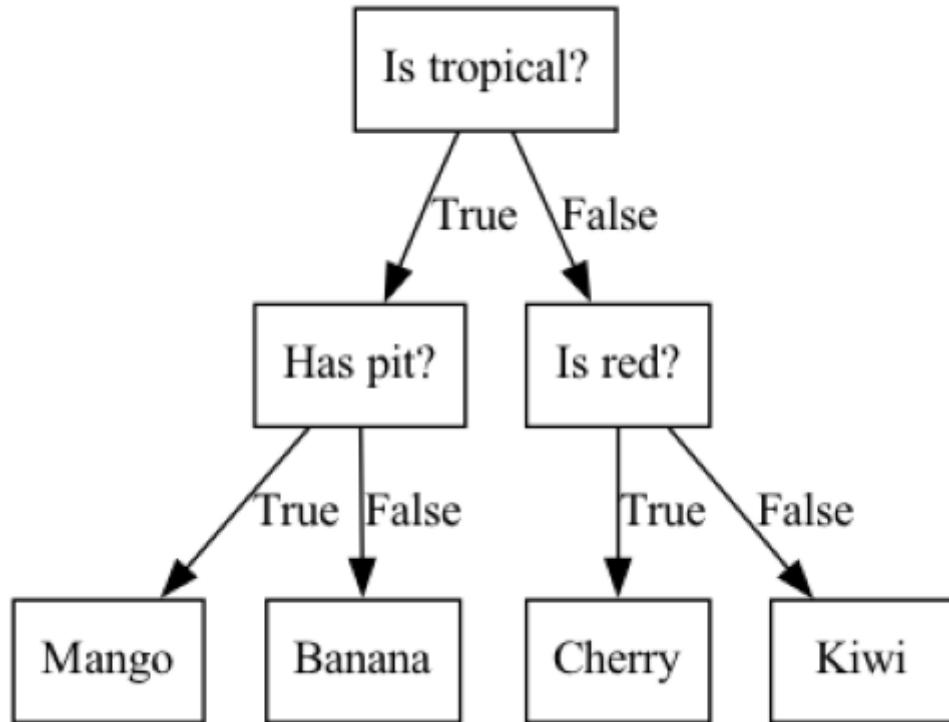
In summary, given a learned tree and a test example, during prediction time,

- Start at the top of the tree. Ask binary questions at each node and follow the appropriate path in the tree. Once you are at a leaf node, you have the prediction.
- Note that the model only considers the features which are in the learned tree and ignores all other features.

How does fit work?

- Decision tree is inspired by [20-questions game](https://en.wikipedia.org/wiki/Twenty_questions) (https://en.wikipedia.org/wiki/Twenty_questions).
- Each node either represents a question or an answer. The terminal nodes (called leaf nodes) represent answers.

In [31]: 1 plot_fruit_tree()



How does fit work?

- Which features are most useful for classification?
- Minimize **impurity** at each question
- Common criteria to minimize impurity: [gini index \(<https://scikit-learn.org/stable/modules/tree.html#classification-criteria>\)](https://scikit-learn.org/stable/modules/tree.html#classification-criteria), information gain, cross entropy

In [36]: 1 `from sklearn.tree import DecisionTreeClassifier`
2
3 `model = DecisionTreeClassifier() # Create a decision tree`
4 `model.fit(X_binary, y) # Fit a decision tree`
5 `display_tree(X_binary.columns, model)`

Out[36]: <graphviz.sources.Source at 0x1879b90f0>

We won't go through **how** it does this - that's CPSC 340. But it's worth noting that it support two types of inputs: 1. Categorical (e.g., Yes/No or more options, as shown in the tree above) 2. Numeric (a number)
In the numeric case, the decision tree algorithm also picks the *threshold*.

Decision trees with continuous features

In [33]:

```
1 X.head()
```

Out[33]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1
0	1		92	93	84	91	92
1	1		94	90	80	83	91
2	0		78	85	83	80	80
3	0		91	94	92	91	89
4	0		77	83	90	92	85

In [51]:

```
1 model = DecisionTreeClassifier()
2 model.fit(X, y)
3 display_tree(X.columns, model)
```

Out[51]:

Decision tree for regression problems

- We can also use decision tree algorithm for regression.
- Instead of gini, we use [some other criteria \(<https://scikit-learn.org/stable/modules/tree.html#mathematical-formulation>\)](https://scikit-learn.org/stable/modules/tree.html#mathematical-formulation) for splitting. A common one is mean squared error (MSE). (More on this in later videos.)
- scikit-learn supports regression using decision trees with `DecisionTreeRegressor`
 - `fit` and `predict` paradigms similar to classification
 - `score` returns somethings called [\$R^2\$ score \(\[https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score\]\(https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score\)\)](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score)
 - The maximum R^2 is 1 for perfect predictions.
 - It can be negative which is very bad (worse than `DummyRegressor`).

In [37]:

```
1 regression_df = pd.read_csv("../data/quiz2-grade-toy-regression.csv")
2 regression_df.head()
```

Out[37]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2
0	1		92	93	84	91	92	90
1	1		94	90	80	83	91	84
2	0		78	85	83	80	80	82
3	0		91	94	92	91	89	92
4	0		77	83	90	92	85	90

In [38]:

```

1 X = regression_df.drop(["quiz2"], axis=1)
2 y = regression_df["quiz2"]
3
4 depth = 2
5 reg_model = DecisionTreeRegressor(max_depth=depth)
6 reg_model.fit(X, y);
7 regression_df["predicted_quiz2"] = reg_model.predict(X)
8 print("R^2 score on the training data: %0.3f\n\n" % (reg_model.score(X,
9 regression_df.head())

```

R² score on the training data: 0.989

Out[38]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2	predicted_quiz2	
0	1		1	92	93	84	91	92	90	90.333333
1	1		0	94	90	80	83	91	84	83.000000
2	0		0	78	85	83	80	80	82	83.000000
3	0		1	91	94	92	91	89	92	92.000000
4	0		1	77	83	90	92	85	90	90.333333

?? True or False?

1. For the decision tree algorithm to work, the feature values must be numeric.
2. For the decision tree algorithm to work, the target values must be numeric.
3. When working with numeric features, the tree will pick the mean as threshold value.

Answer here: <https://www.menti.com/albuty3ge2d4> (<https://www.menti.com/albuty3ge2d4>)

Break (5 min)



More terminology [video (<https://youtu.be/KEtsfXn4w2E>)]

- Parameters and hyperparameters
- Decision boundary

Check out [the accompanying video](https://youtu.be/KEtsfXn4w2E) (<https://youtu.be/KEtsfXn4w2E>) on this material.

Parameters

- The decision tree algorithm primarily learns two things:
 - the best feature to split on
 - the threshold for the feature to split on at each node

- These are called **parameters** of the decision tree model.
- When predicting on new examples, we need parameters of the model.

```
In [39]: 1 classification_df = pd.read_csv("../data/quizz2-grade-toy-classification"
2 X = classification_df.drop(columns=["quizz2"])
3 y = classification_df["quizz2"]
4 model = DecisionTreeClassifier()
5 model.fit(X, y);
```

```
In [40]: 1 display_tree(X.columns, model, counts=True)
```

Out[40]: <graphviz.sources.Source at 0x1878a5120>

- With the default setting, the nodes are expanded until all leaves are "pure" (or further splits do not improve the classification).

- The decision tree is creating very specific rules, based on just one example from the data.
- Is it possible to control the learning in any way?
 - Yes! One way to do it is by controlling the **depth** of the tree, which is the length of the longest path from the tree root to a leaf.

Decision tree with `max_depth=1`

Decision stump : A decision tree with only one split (depth=1) is called a **decision stump**.

```
In [42]: 1 model = DecisionTreeClassifier(max_depth=1)
2 model.fit(X, y)
3 display_tree(X.columns, model, counts=True)
```

Out[42]: <graphviz.sources.Source at 0x1877cb6d0>

`max_depth` is a **hyperparameter** of `DecisionTreeClassifier` .

Decision tree with `max_depth=3`

```
In [41]: 1 model = DecisionTreeClassifier(
2     max_depth=3
3 ) # Let's try another value for the hyperparameter
4 model.fit(X, y)
5 display_tree(X.columns, model, counts=True)
```

Out[41]: <graphviz.sources.Source at 0x18766f970>

Parameters and hyperparameters: Summary

Parameters : When you call `fit`, a bunch of values get set, like the features to split on and split thresholds. These are called **parameters**. These are learned by the algorithm from the data during training. We need them during prediction time.

Hyperparameters : Even before calling `fit` on a specific data set, we can set some "knobs" that control the learning. These are called **hyperparameters**. These are specified based on: expert knowledge, heuristics, or systematic/automated optimization (more on this in the coming lectures).

In `sklearn` hyperparameters are set in the constructor.

Above we looked at the `max_depth` hyperparameter. Some other commonly used hyperparameters of decision tree are:

- `min_samples_split`
- `min_samples_leaf`
- `max_leaf_nodes`

See [the documentation \(<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>\)](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html) for other hyperparameters of a tree.

Decision boundary

What do we do with learned models? So far we have been using them to predict the class of a new instance. Another way to think about them is to ask: what sort of test examples will the model classify as positive, and what sort will it classify as negative?

Example 1: quiz 2 grade prediction

For visualization purposes, let's consider a subset of the data with only two features.

```
In [ ]: 1 X_subset = X[["lab4", "quiz1"]]
2 X_subset.head()
```

Decision boundary for `max_depth=1`

In []:

```

1 depth = 1 # decision stump
2 model = DecisionTreeClassifier(max_depth=depth)
3 model.fit(X_subset.to_numpy(), y)
4 plot_tree_decision_boundary_and_tree(
5     model, X_subset, y, x_label="lab4", y_label="quiz1"
6 )

```

We assume geometric view of the data. Here, the red region corresponds to "not A+" class and blue region corresponds to "A+" class. And there is a line separating the red region and the blue region which is called the **decision boundary** of the model. Different models have different kinds of decision boundaries. In decision tree models, when we are working with only two features, the decision boundary is made up of horizontal and vertical lines. In the example above, the decision boundary is created by asking one question `lab4 <= 84.5`.

Decision boundary for `max_depth=2`

In []:

```

1 model = DecisionTreeClassifier(max_depth=2)
2 model.fit(X_subset.to_numpy(), y)
3 plot_tree_decision_boundary_and_tree(
4     model, X_subset, y, x_label="lab4", y_label="quiz1"
5 )

```

The decision boundary, i.e., the model gets a bit more complicated.

Decision boundary for `max_depth=5`

In []:

```

1 model = DecisionTreeClassifier(max_depth=5)
2 model.fit(X_subset.to_numpy(), y)
3 plot_tree_decision_boundary_and_tree(
4     model, X_subset, y, x_label="lab4", y_label="quiz1"
5 )

```

The decision boundary, i.e., the model gets even more complicated with `max_depth=5`.

Example 2: Predicting country using the longitude and latitude

Imagine that you are given longitude and latitude of some border cities of USA and Canada along with which country they belong to. Using this training data, you are supposed to come up with a classification model to predict whether a given longitude and latitude combination is in the USA or Canada.

In [1]:

```
1 # US Canada cities data
2 df = pd.read_csv("../data/canada_usa_cities.csv")
3 df
```

```
--  
NameError
```

```
Traceback (most recent call last)
```

```
t)  
Cell In[1], line 2  
    1 # US Canada cities data  
----> 2 df = pd.read_csv("../data/canada_usa_cities.csv")  
    3 df
```

```
NameError: name 'pd' is not defined
```

In []:

```
1 X = df[["longitude", "latitude"]]
```

In []:

```
1 y = df["country"]
```

In []:

```
1 mglearn.discrete_scatter(X.iloc[:, 0], X.iloc[:, 1], y)
2 plt.xlabel("longitude")
3 plt.ylabel("latitude");
```

Real boundary between Canada and USA

In real life we know what's the boundary between USA and Canada.



```
In [ ]: 1 model = DecisionTreeClassifier(max_depth=1)
2 model.fit(X.to_numpy(), y)
3 plot_tree_decision_boundary_and_tree(
4     model,
5     X,
6     y,
7     height=6,
8     width=16,
9     eps=10,
10    x_label="longitude",
11    y_label="latitude",
12 )
```

```
In [ ]: 1 model = DecisionTreeClassifier(max_depth=2)
2 model.fit(X.to_numpy(), y)
3 plot_tree_decision_boundary_and_tree(
4     model,
5     X,
6     y,
7     height=6,
8     width=16,
9     eps=10,
10    x_label="longitude",
11    y_label="latitude",
12 )
```

Practice exercises

- If you want more practice, check out module 2 in [this online course](https://ml-learn.mds.ubc.ca/en/module2) (<https://ml-learn.mds.ubc.ca/en/module2>). All the sections **without** video or notes symbol are exercises.

If all of you are working on the exercises, especially coding exercises, at the same time, you might have to wait for the real-time feedback for a long time or you might even get an error. There is no solution for this other than waiting for a while and trying it again.

Some background on [the online course](https://ml-learn.mds.ubc.ca/en/) (<https://ml-learn.mds.ubc.ca/en/>) above: This course is designed by Hayley Boyce, Mike Gelbart, and Varada Kolhatkar. It'll be a great resource at the beginning of this class, as it give you a chance to practice what we learn and the framework will provide you real-time feedback.

Final comments and summary

What did we learn today?

- There is a lot of terminology and jargon used in ML. Some of the basic terminology includes:
 - Features, target, examples, training
 - Supervised vs. Unsupervised machine learning
 - Classification and regression
 - Accuracy and error
 - Parameters and hyperparameters
 - Decision boundary
- Baselines and steps to train a supervised machine learning model
 - Baselines serve as reference points in ML workflow.
- Decision trees
 - are models that make predictions by sequentially looking at features and checking whether they are above/below a threshold
 - learn a hierarchy of if/else questions, similar to questions you might ask in a 20-questions game.
 - learn axis-aligned decision boundaries (vertical and horizontal lines with 2 features)
 - One way to control the complexity of decision tree models is by using the depth hyperparameter (`max_depth` in `sklearn`).



In []:

1

CPSC 330

Applied Machine Learning

Lecture 3: Machine Learning Fundamentals

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

```
In [1]: 1 # import the libraries
2 import os
3 import sys
4
5 import graphviz
6 import IPython
7 import matplotlib.pyplot as plt
8 import numpy as np
9 import pandas as pd
10 from IPython.display import HTML
11 from sklearn.model_selection import train_test_split
12
13 sys.path.append("../code/.")
14 from plotting_functions import *
15
16 # Classifiers
17 from sklearn.tree import DecisionTreeClassifier, export_graphviz
18 from utils import *
19
20 %matplotlib inline
21
22 pd.set_option("display.max_colwidth", 200)
```

Learning outcomes

From this lecture, you will be able to:

- Understand model complexity and generalization
- Estimate the generalization error with data splitting

- Understand and use cross-validation
- Understand overfitting, underfitting, and related ML practice

Details:

Understand model complexity and generalization:

- explain how decision boundaries change with the `max_depth` hyperparameter;
- explain the concept of generalization;

Estimate the generalization error with data splitting

- appropriately split a dataset into train and test sets using `train_test_split` function;
- explain the difference between train, validation, test, and "deployment" data;
- identify the difference between training error, validation error, and test error;

Understand and use cross-validation

- explain cross-validation and use `cross_val_score` and `cross_validate` to calculate cross-validation error;

Understand overfitting, underfitting, and related ML practice

- recognize overfitting and/or underfitting by looking at train and test scores;
- explain why it is generally not possible to get a perfect test score (zero test error) on a supervised learning problem;
- describe the fundamental tradeoff between training score and the train-test gap;
- state the golden rule;
- start to build a standard recipe for supervised learning: train/test split, hyperparameter tuning with cross-validation, test on test set.

Announcements

- For assignments, setup, etc: use material from this year! Here: <https://github.com/UBC-CS/cpsc330-2022W2> (<https://github.com/UBC-CS/cpsc330-2022W2>). Links to older content are there if you want to explore, but not for deliverables.
- hw2 released. (Due next week Monday Jan 23 at 11:59pm.)
 - You are allowed to submit in pairs.
 - If you need to find a partner to work with, check out <https://piazza.com/class/lcg06c2ncl06el/post/5> (<https://piazza.com/class/lcg06c2ncl06el/post/5>).
- Advice on keeping up with the material
 - Practice! You will find some practice questions [here](https://ml-learn.mds.ubc.ca/) (<https://ml-learn.mds.ubc.ca/>).
 - Start early on homework assignments.
- Last day to withdraw without a W standing: January 23rd, 2022

Homework check in: <https://www.menti.com/als14k4vqbzx> (<https://www.menti.com/als14k4vqbzx>)

Recap

Last week, we introduced the following concepts:

- General idea of what Machine Learning is and its applications
- Basic terminology (such as features vs. target, unsupervised vs. supervised learning, classification vs. regression)
- Why is it a good idea to train a baseline classifier and how to do it (this included the basic steps of training using `scikit-learn`)
- Decision trees
- Parameters (e.g. decision tree rules) and hyperparameters (e.g., maximum tree depth)
- Decision boundary (today)

Pre-lecture videos

You were asked to watch 2 videos before coming to class, one on [generalization](https://youtu.be/iS2hsRRlc2M) (<https://youtu.be/iS2hsRRlc2M>), one on [data splitting](https://youtu.be/h2AEobwcUQw) (<https://youtu.be/h2AEobwcUQw>).

We will revisit the first one to discuss dicision boundaries, and quickly mention the second.

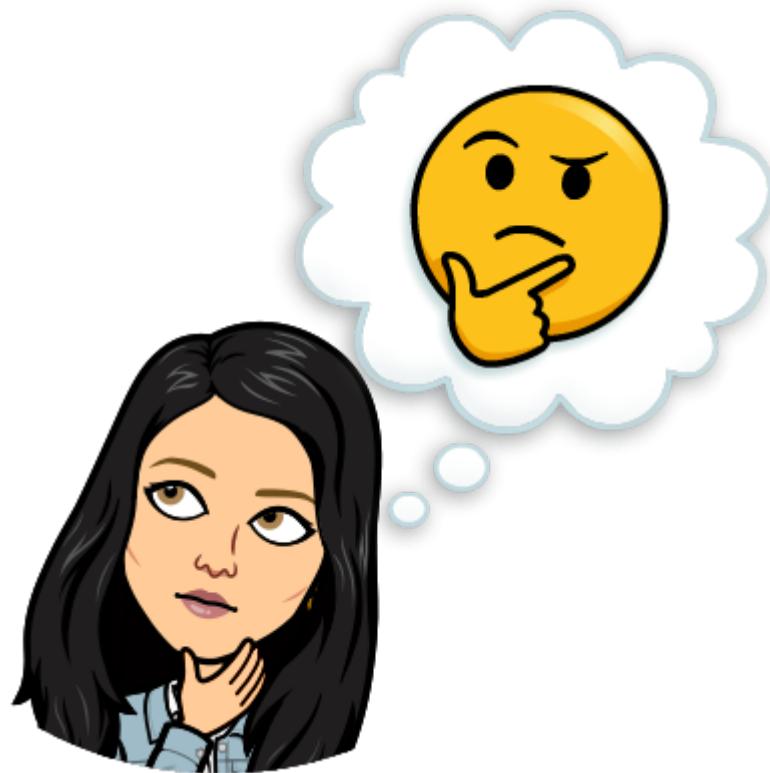
Generalization [[video \(https://youtu.be/iS2hsRRlc2M\)](https://youtu.be/iS2hsRRlc2M)]

Big picture and motivation

In machine learning, we want to glean information from labeled data so that we can label **new unlabeled** data.

For example, suppose we want to build a spam filtering system. We will take a large number of spam/non-spam messages from the past, learn patterns associated with spam/non-spam from them, and predict whether **a new incoming message** in someone's inbox is spam or non-spam based on these patterns.

So we want to learn from the past but ultimately we want to apply it on the future email messages.



How can we generalize from what we've seen to what we haven't seen?

In this lecture, we'll see how machine learning tackles this question.

Model complexity and training error

Let's examine the **decision boundary** of tree classifiers, and how it changes for different tree depths, to visualize what sort of examples will be classified as positive and negative.

```
In [2]: 1 # Toy quiz2 grade data
2 classification_df = pd.read_csv("../data/quiz2-grade-toy-classification")
3 classification_df.head(10)
```

Out[2]:

	ml_experience	class_attendance	lab1	lab2	lab3	lab4	quiz1	quiz2	
0	1		1	92	93	84	91	92	A+
1	1		0	94	90	80	83	91	not A+
2	0		0	78	85	83	80	80	not A+
3	0		1	91	94	92	91	89	A+
4	0		1	77	83	90	92	85	A+
5	1		0	70	73	68	74	71	not A+
6	1		0	80	88	89	88	91	A+
7	0		1	95	93	69	79	75	not A+
8	0		0	97	90	94	99	80	not A+
9	1		1	95	95	94	94	85	not A+

```
In [3]: 1 X = classification_df.drop(["quiz2"], axis=1)
2 y = classification_df["quiz2"]
3
4 X_subset = X[["lab4", "quiz1"]] # Let's consider a subset of the data
5 X_subset.head()
```

Out[3]:

	lab4	quiz1
0	91	92
1	83	91
2	80	80
3	91	89
4	92	85

```
In [ ]: 1 X_subset = X[["lab4", "quiz1"]] # Let's consider a subset of the data
2 X_subset.head()
```

In the following model (decision stump), this decision boundary is created by asking one question.

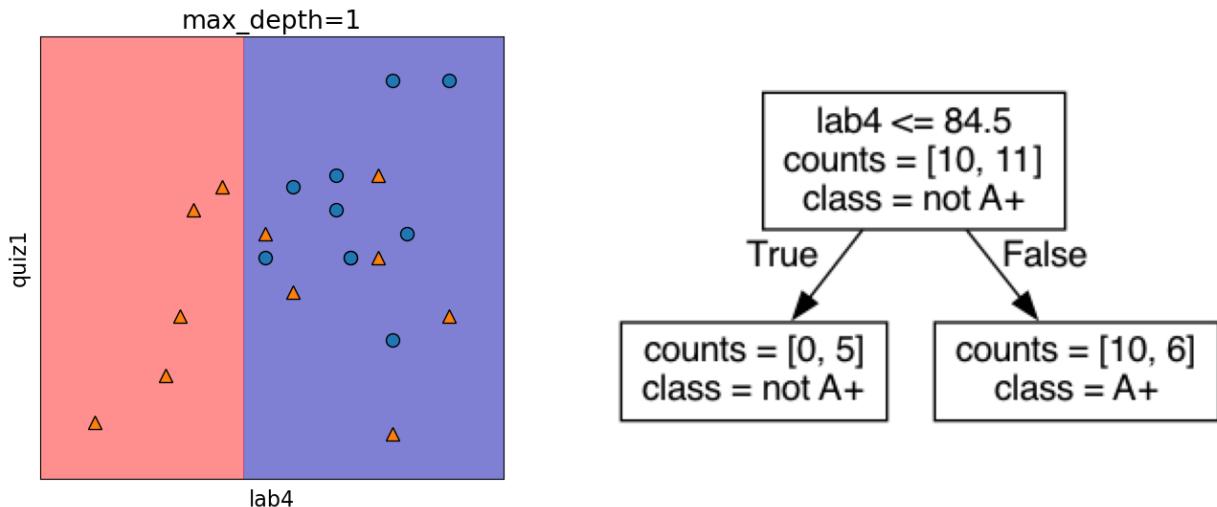
In [4]:

```

1 depth = 1
2 model = DecisionTreeClassifier(max_depth=depth)
3 model.fit(X_subset.to_numpy(), y)
4 model.score(X_subset.to_numpy(), y)
5 print("Error: %0.3f" % (1 - model.score(X_subset.to_numpy(), y)))
6 plot_tree_decision_boundary_and_tree(model, X_subset, y, x_label="lab4")

```

Error: 0.286



In []:

1

In the following model, this decision boundary is created by asking two questions.

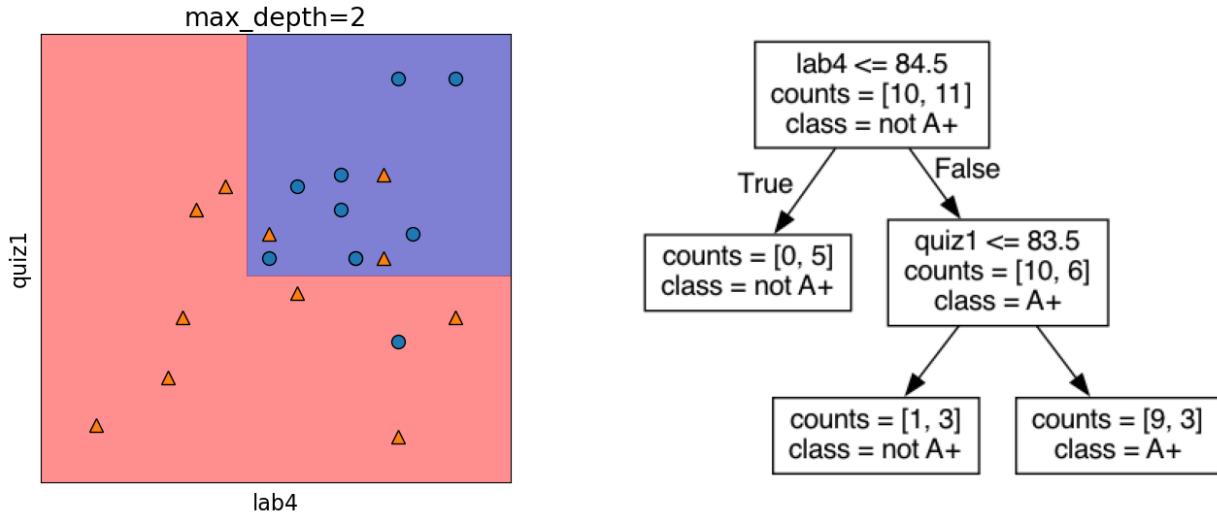
In [5]:

```

1 depth = 2
2 model = DecisionTreeClassifier(max_depth=depth)
3 model.fit(X_subset.to_numpy(), y)
4 model.score(X_subset.to_numpy(), y)
5 print("Error: %0.3f" % (1 - model.score(X_subset.to_numpy(), y)))
6 plot_tree_decision_boundary_and_tree(
    model, X_subset, y, x_label="lab4", y_label="quiz1"
8 )

```

Error: 0.190



Let's look at the decision boundary with depth = 4.

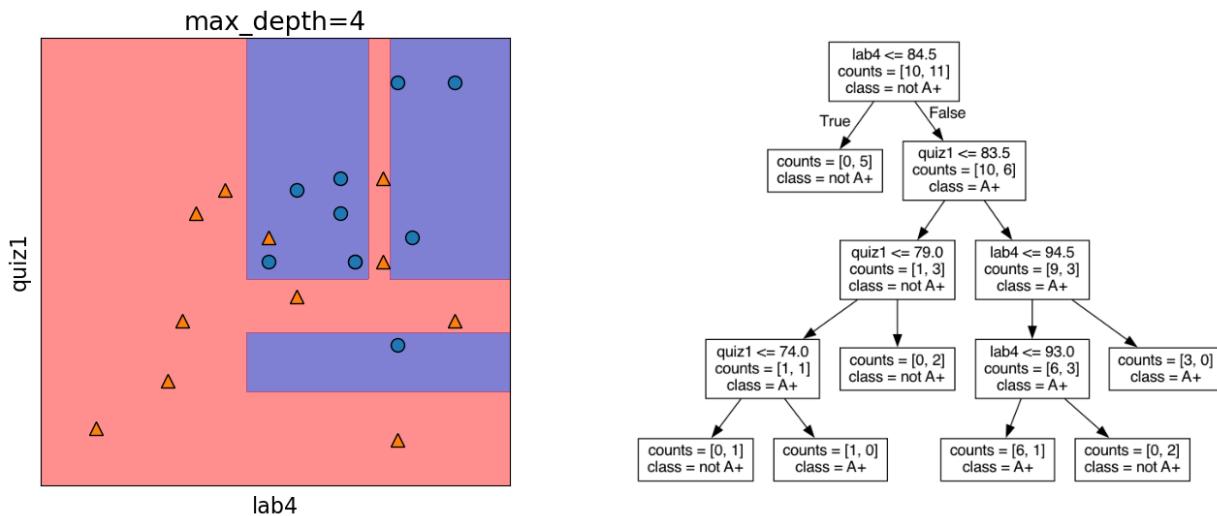
In [6]:

```

1 depth = 4
2 model = DecisionTreeClassifier(max_depth=depth)
3 model.fit(X_subset.to_numpy(), y)
4 model.score(X_subset.to_numpy(), y)
5 print("Error: %0.3f" % (1 - model.score(X_subset.to_numpy(), y)))
6 plot_tree_decision_boundary_and_tree(
    model, X_subset, y, x_label="lab4", y_label="quiz1"
8 )

```

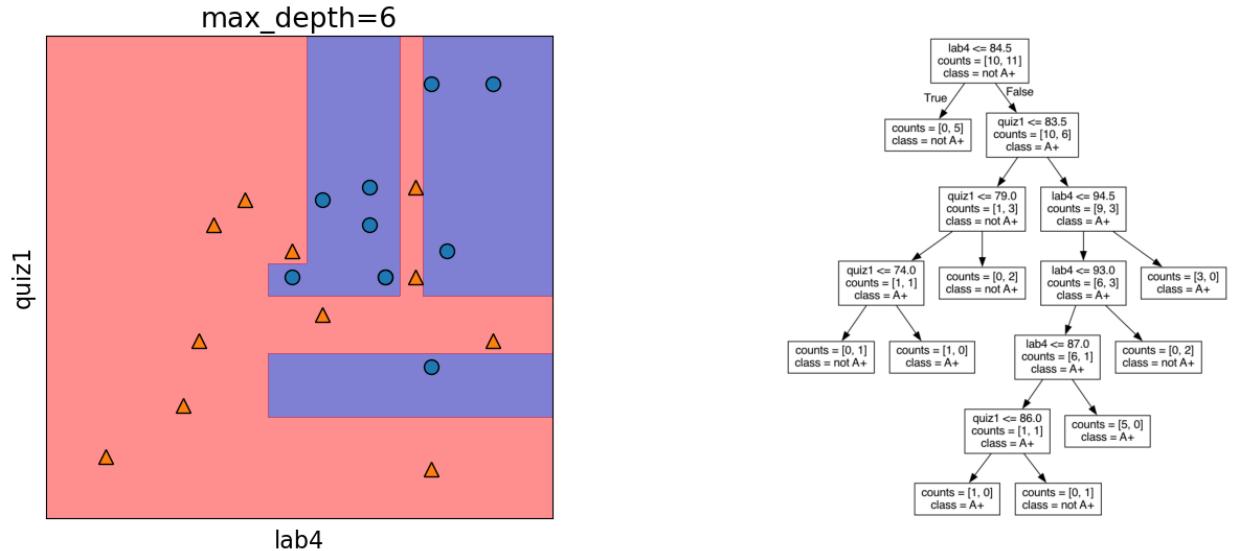
Error: 0.048



Let's look at the decision boundary with depth = 6.

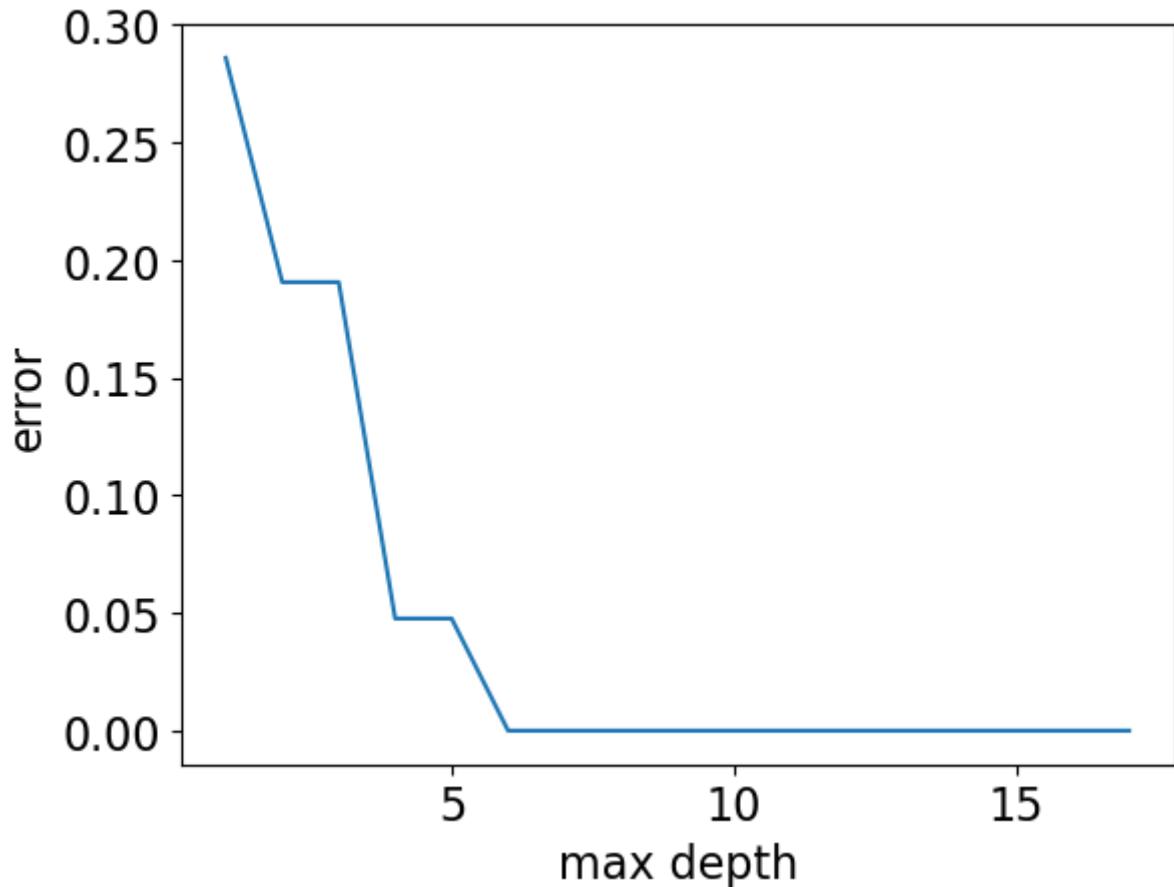
```
In [7]: 1 depth = 6
2 model = DecisionTreeClassifier(max_depth=depth)
3 model.fit(X_subset.to_numpy(), y)
4 model.score(X_subset.to_numpy(), y)
5 print("Error:  %0.3f" % (1 - model.score(X_subset.to_numpy(), y)))
6 plot_tree_decision_boundary_and_tree(
7     model, X_subset, y, x_label="lab4", y_label="quiz1"
8 )
```

Error: 0.000



In [8]:

```
1 max_depths = np.arange(1, 18)
2 errors = []
3 for max_depth in max_depths:
4     error = 1 - DecisionTreeClassifier(max_depth=max_depth).fit(X_subset,
5         X_subset, y
6     )
7     errors.append(error)
8 plt.plot(max_depths, errors)
9 plt.xlabel("max depth")
10 plt.ylabel("error");
```



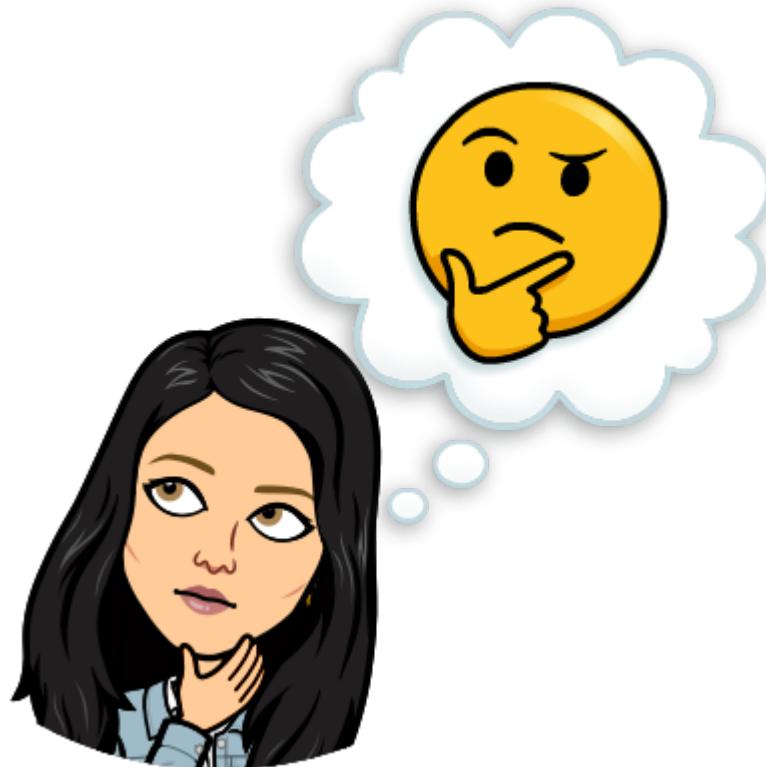
- Our model has 0% error for depths ≥ 6 !!
- But it's also becoming more and more specific and sensitive to the training data.
- Is it good or bad?

Although the plot above (complexity hyperparameter vs error) is more popular, we could also look at the same plot flip the y-axis, i.e., consider accuracy instead of error.

In []:

```
1 max_depths = np.arange(1, 18)
2 accuracies = []
3 for max_depth in max_depths:
4     accuracy = (
5         DecisionTreeClassifier(max_depth=max_depth).fit(X_subset, y).sc
6     )
7     accuracies.append(accuracy)
8 plt.plot(max_depths, accuracies)
9 plt.xlabel("max depth")
10 plt.ylabel("accuracy");
```

🤔 Eva's questions



At this point Eva is wondering about the following questions.

- How to pick the best depth?
- How can we make sure that the model we have built would do reasonably well on new data in the wild when it's deployed?
- Which of the following rules learned by the decision tree algorithm are likely to generalize better to new data?

Rule 1: If class_attendance == 1 then grade is A+.

Rule 2: If lab3 > 83.5 and quiz1 <= 83.5 and lab2 <= 88 then quiz2 grade is A+

Generalization: Fundamental goal of ML

To generalize beyond what we see in the training examples

We only have access to limited amount of training data and we want to learn a mapping function which would predict targets reasonably well for examples beyond this training data.

- Example: Imagine that a learner sees the following images and corresponding labels.

Generalizing to unseen data

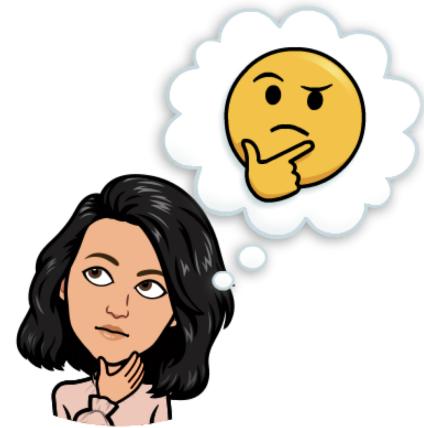
- Now the learner is presented with new images (1 to 4) for prediction.
- What prediction would you expect for each image?

Training data

	CAT
	CAT
	DOG
	DOG

New examples
for prediction

1		?
2		?
3		?
4		?



- Goal: We want the learner to be able to generalize beyond what it has seen in the training data.
- But these new examples should be representative of the training data. That is they should have the same characteristics as the training data.
- In this example, we would like the learner to be able to predict labels for test examples 1, 2, and 3 accurately. Although 2, 3 don't exactly occur in the training data, they are very much similar to the images in the training data. That said, is it fair to expect the learner to label image 4 correctly?

Training error vs. Generalization error

- Given a model M , in ML, people usually talk about two kinds of errors of M .
 - Error on the training data: $\text{error}_{\text{training}}(M)$
 - Error on the entire distribution D of data: $\text{error}_D(M)$
- We are interested in the error on the entire distribution
 - ... But we do not have access to the entire distribution 😞

Data Splitting [video (<https://youtu.be/h2AEobwcUQw>)]

How to approximate generalization error?

A common way is **data splitting**.

- Keep aside some randomly selected portion from the training data.
- `fit` (train) a model on the training portion only.
- `score` (assess) the trained model on this set aside data to get a sense of how well the model would be able to generalize.
- Pretend that the kept aside data is representative of the real distribution D of data.



```
In [9]: 1 # scikit-learn train_test_split
2 url = "https://scikit-learn.org/stable/modules/generated/sklearn.model_
3 IPython.display.IFrame(url, width=1000, height=800)
```

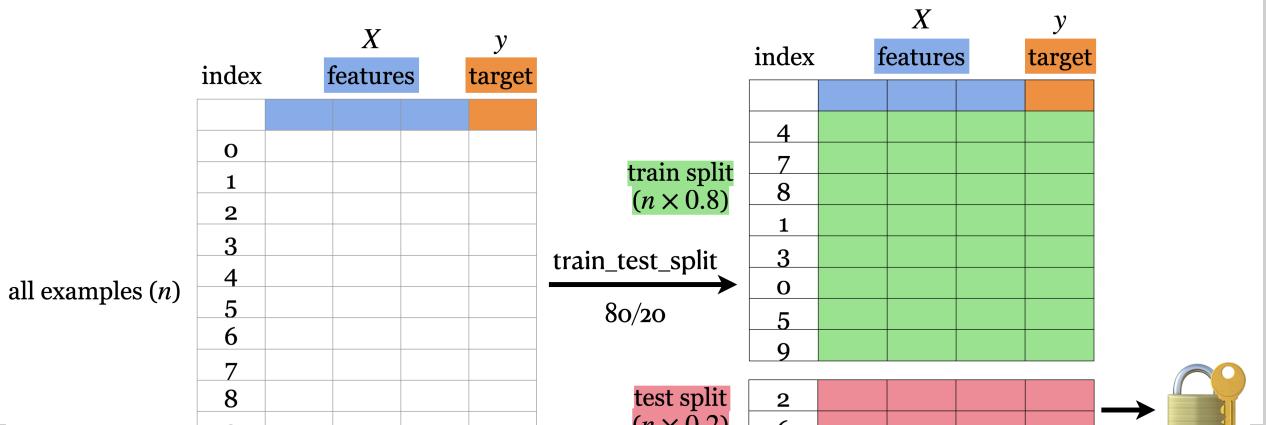
Out[9]:

- We can pass `x` and `y` or a dataframe with both `x` and `y` in it.
- We can also specify the train or test split sizes.

Simple train/test split

- The picture shows an 80%-20% split of a toy dataset with 10 examples.

- The data is shuffled before splitting.
- Usually when we do machine learning we split the data before doing anything and put the test data in an imaginary chest lock.



```
In [10]: 1 # Let's demonstrate this with the canada usa cities data
          2 # The data is available in the data directory
          3 df = pd.read_csv("../data/canada_usa_cities.csv")
          4 X = df.drop(columns=["country"])
          5 y = df["country"]
```

```
In [ ]: 1 X
```

```
In [ ]: 1 y
```

In [11]:

```

1 from sklearn.model_selection import train_test_split
2
3 X_train, X_test, y_train, y_test = train_test_split(
4     X, y, test_size=0.2, random_state=123
5 ) # 80%-20% train test split on X and y
6
7 # Print shapes
8 shape_dict = {
9     "Data portion": ["X", "y", "X_train", "y_train", "X_test", "y_test"]
10    "Shape": [
11        X.shape,
12        y.shape,
13        X_train.shape,
14        y_train.shape,
15        X_test.shape,
16        y_test.shape,
17    ],
18 }
19
20 shape_df = pd.DataFrame(shape_dict)
21 HTML(shape_df.to_html(index=False))

```

Out[11]:

	Data portion	Shape
X	(209, 2)	
y	(209,)	
X_train	(167, 2)	
y_train	(167,)	
X_test	(42, 2)	
y_test	(42,)	

Creating `train_df` and `test_df`

- Sometimes we want to keep the target in the train split for EDA or for visualization.

In []:

```

1 train_df, test_df = train_test_split(
2     df, test_size=0.2, random_state=123
3 ) # 80%-20% train test split on df
4 X_train, y_train = train_df.drop(columns=["country"]), train_df["country"]
5 X_test, y_test = test_df.drop(columns=["country"]), test_df["country"]
6 train_df.head()

```

In []:

```

1 mglearn.discrete_scatter(X.iloc[:, 0], X.iloc[:, 1], y, s=12)
2 plt.xlabel("longitude")
3 plt.ylabel("latitude");

```

```
In [12]: 1 model = DecisionTreeClassifier()  
2 model.fit(X_train.to_numpy(), y_train)  
3 display_tree(X_train.columns, model)
```

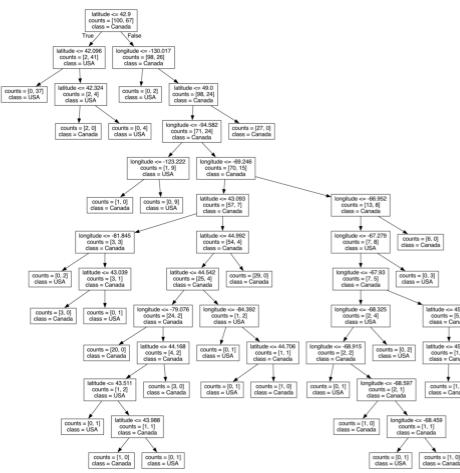
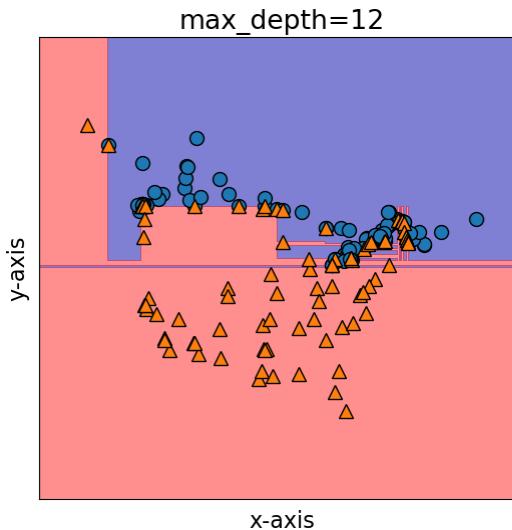
Out[12]: <graphviz.sources.Source at 0x182c5fd90>

Let's examine the train and test accuracies with the split now.

```
In [13]: 1 print("Train accuracy:  %0.3f" % model.score(X_train.to_numpy(), y_train))
          2 print("Test accuracy:   %0.3f" % model.score(X_test.to_numpy(), y_test))
```

Train accuracy: 1.000
Test accuracy: 0.738

```
In [14]: 1 plot_tree_decision_boundary_and_tree(model, x, y, height=6, width=16, e
```

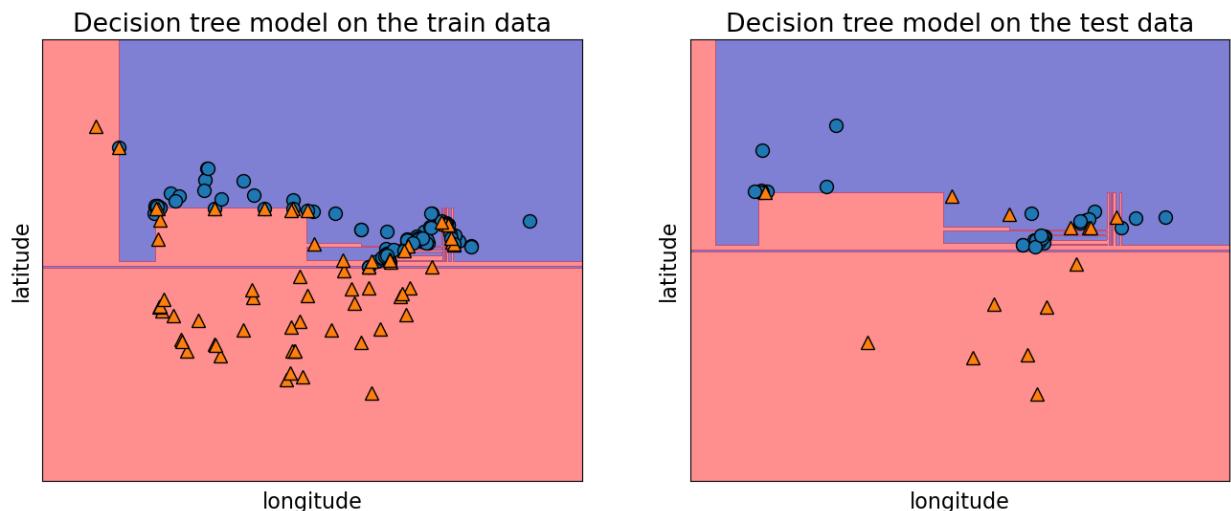


In [15]:

```

1 fig, ax = plt.subplots(1, 2, figsize=(16, 6), subplot_kw={"xticks": ()},
2 plot_tree_decision_boundary(
3     model,
4     X_train,
5     y_train,
6     eps=10,
7     x_label="longitude",
8     y_label="latitude",
9     ax=ax[0],
10    title="Decision tree model on the train data",
11 )
12 plot_tree_decision_boundary(
13     model,
14     X_test,
15     y_test,
16     eps=10,
17     x_label="longitude",
18     y_label="latitude",
19     ax=ax[1],
20    title="Decision tree model on the test data",
21 )

```



What could we do to make the model generalize better to unseen data?

- Useful arguments of `train_test_split`:
 - `test_size`
 - `train_size`
 - `random_state`

`test_size`, `train_size` arguments

- Let's us specify how we want to split the data.
- We can specify either of the two. See the documentation [here](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html).

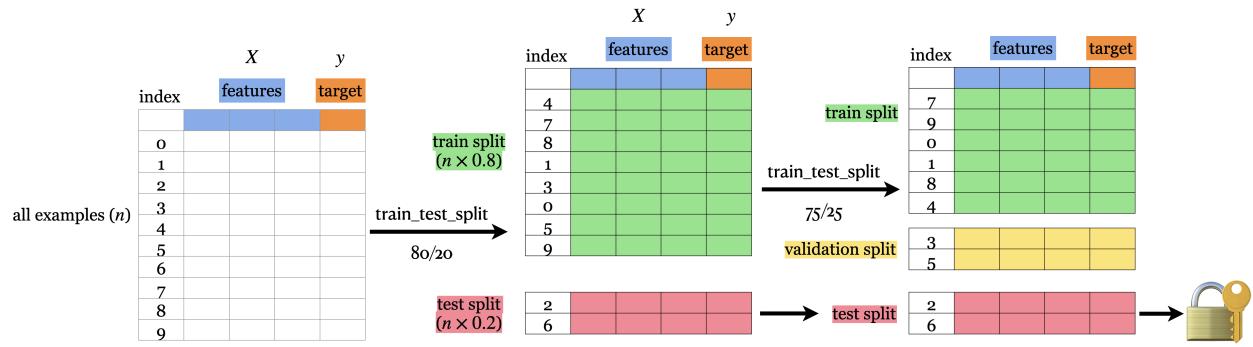
- There is no hard and fast rule on what split sizes should we use.
 - It depends upon how much data is available to you.
- Some common splits are 90/10, 80/20, 70/30 (training/test).
- In the above example, we used 80/20 split.

random_state argument

- The data is shuffled before splitting which is crucial step. (You will explore this in the lab.)
- The `random_state` argument controls this shuffling.
- In the example above we used `random_state=123`. If you run this notebook with the same `random_state` it should give you exactly the same split.
 - Useful when you want reproducible results.

Train/validation/test split

- Some of you may have heard of "validation" data.
- Sometimes it's a good idea to have a separate data for hyperparameter tuning.



- We will try to use "validation" to refer to data where we have access to the target values.
 - But, unlike the training data, we only use this for hyperparameter tuning and model assessment; we don't pass these into `fit`.
- We will try to use "test" to refer to data where we have access to the target values
 - But, unlike training and validation data, we neither use it in training nor hyperparameter optimization.
 - We only use it **once** to evaluate the performance of the best performing model on the validation set.
 - We lock it in a "vault" until we're ready to evaluate.

Note that there isn't good consensus on the terminology of what is validation and what is test.

Validation data is also referred to as **development data** or **dev set** for short.

"Deployment" data

- After we build and finalize a model, we deploy it, and then the model deals with the data in the wild.
- We will use "deployment" to refer to this data, where we do **not** have access to the target values.
- Deployment error is what we *really* care about.
- We use validation and test errors as proxies for deployment error, and we hope they are similar.
- So, if our model does well on the validation and test data, we hope it will do well on deployment data.

Summary of train, validation, test, and deployment data

	fit	score	predict
Train	✓	✓	✓
Validation		✓	✓
Test		once	once
Deployment			✓

You can typically expect $E_{train} < E_{validation} < E_{test} < E_{deployment}$.

?? Questions on generalization and data splitting

1. A decision tree model with no depth is likely to perform very well on the deployment data.
2. Data splitting helps us generalize our model better.
3. Deployment data is used at the very end and only scored once.
4. Validation data could be used for hyperparameter optimization.

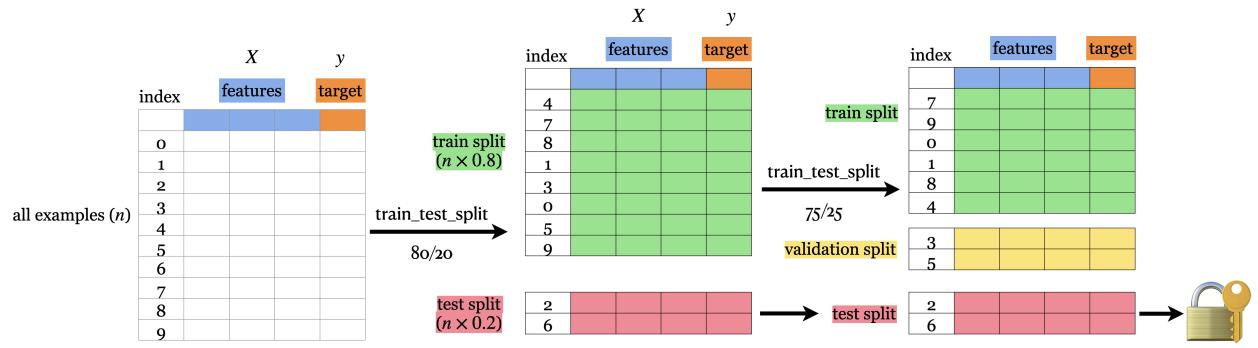
1. False
2. False. Data splitting helps us **assess** how well our model would generalize.
3. False. You cannot score on the deployment data as there are no labels available.
4. True

1. Why you can typically expect $E_{train} < E_{validation} < E_{test} < E_{deployment}$.
2. Discuss the consequences of not shuffling before splitting the data in `train_test_split`.

Cross-validation [video (<https://youtu.be/4cv8VYonepA>)]

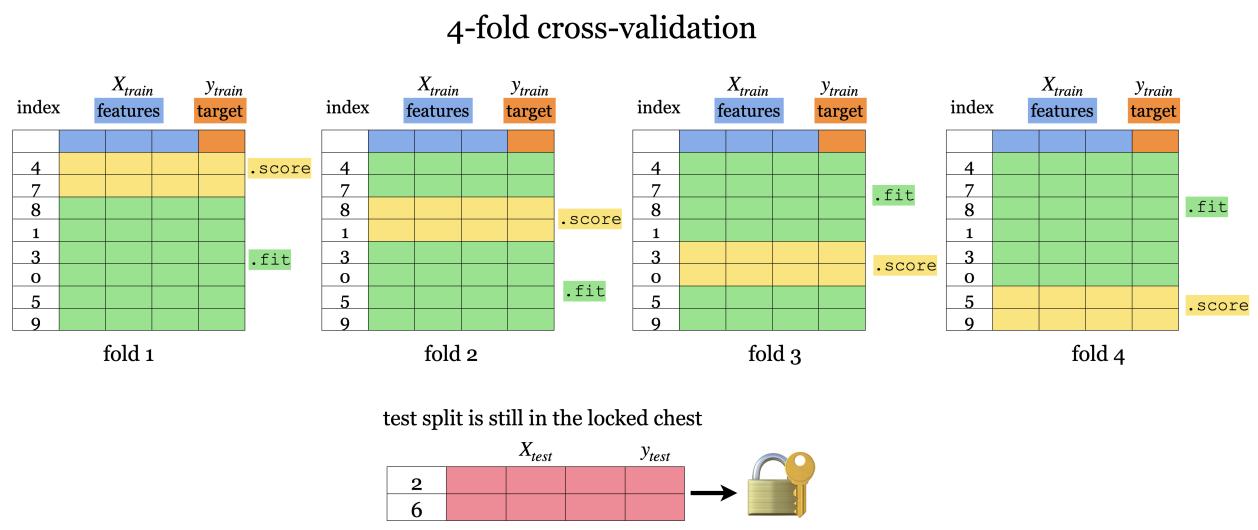
Problems with single train/validation split

- Only using a portion of your data for training and only a portion for validation.
- If your dataset is small you might end up with a tiny training and/or validation set.
- You might be unlucky with your splits such that they don't align well or don't well represent your test data.



Cross-validation to the rescue!!

- Cross-validation provides a solution to this problem.
- Split the data into k folds ($k > 2$, often $k = 10$). In the picture below $k = 4$.
- Each "fold" gets a turn at being the validation set.
- Note that cross-validation doesn't shuffle the data; it's done in `train_test_split`.



- Each fold gives a score and we usually average our k results.
- It's better to examine the variation in the scores across folds.
- Gives a more **robust** measure of error on unseen data.

Cross-validation using `scikit-learn`

```
In [16]: 1 # Let's demonstrate this with the canada usa cities data
          2 # The data is available in the data directory
          3 df = pd.read_csv("../data/canada_usa_cities.csv")
          4 X = df.drop(columns=["country"])
          5 y = df["country"]
```

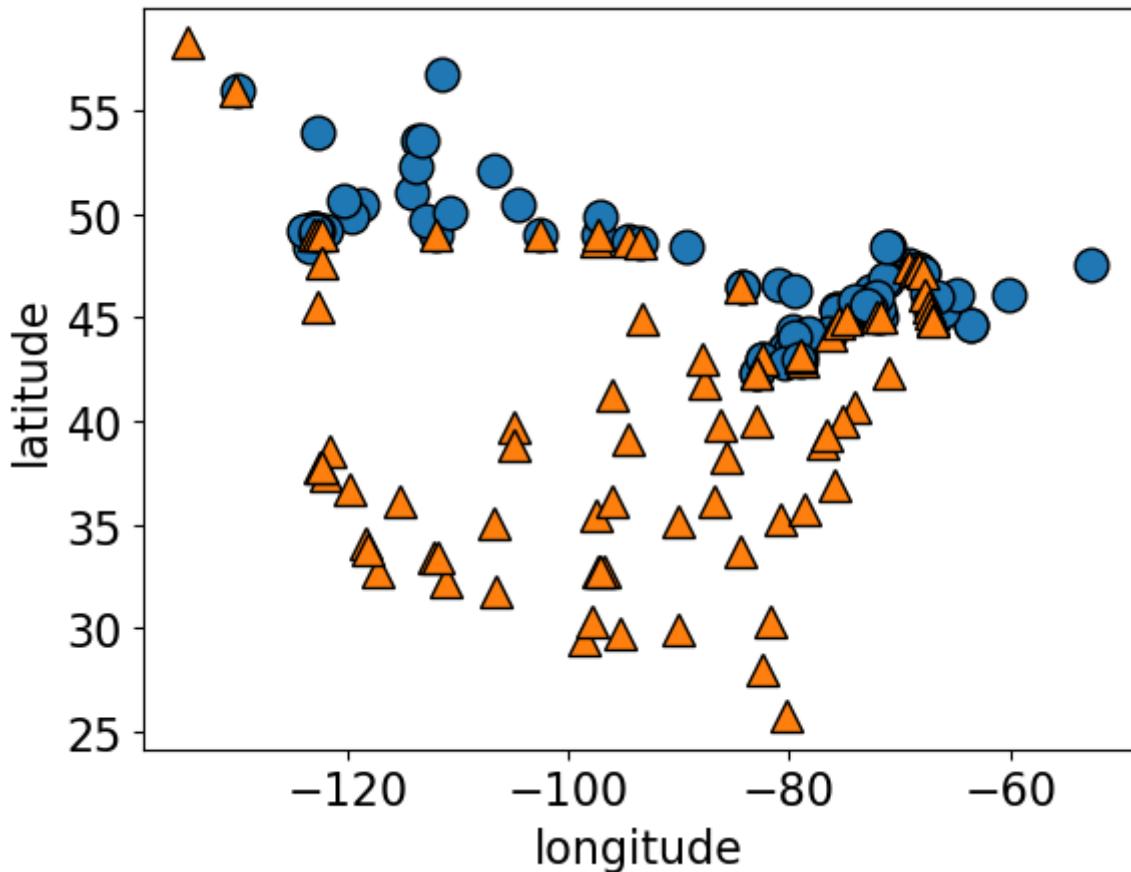
```
In [17]: 1 X
```

Out[17]:

	longitude	latitude
0	-130.0437	55.9773
1	-134.4197	58.3019
2	-123.0780	48.9854
3	-122.7436	48.9881
4	-122.2691	48.9951
...
204	-72.7218	45.3990
205	-66.6458	45.9664
206	-79.2506	42.9931
207	-72.9406	45.6275
208	-79.4608	46.3092

209 rows × 2 columns

```
In [18]: 1 mglearn.discrete_scatter(X.iloc[:, 0], X.iloc[:, 1], y, s=12)
2 plt.xlabel("longitude")
3 plt.ylabel("latitude");
```



```
In [19]: 1 train_df, test_df = train_test_split(
2     df, test_size=0.2, random_state=123
3 ) # 80%-20% train test split on df
4 X_train, y_train = train_df.drop(columns=["country"]), train_df["country"]
5 X_test, y_test = test_df.drop(columns=["country"]), test_df["country"]
```

`cross_val_score`

```
In [20]: 1 from sklearn.model_selection import cross_val_score, cross_validate
```

```
In [21]: 1 model = DecisionTreeClassifier(max_depth=4)
2 cv_scores = cross_val_score(model, X_train, y_train, cv=10)
3 cv_scores
```

```
Out[21]: array([0.76470588, 0.82352941, 0.70588235, 0.94117647, 0.82352941,
 0.82352941, 0.70588235, 0.9375 , 0.9375 , 0.9375 ])
```

```
In [22]: 1 print(f"Average cross-validation score = {np.mean(cv_scores):.2f}")
2 print(f"Standard deviation of cross-validation score = {np.std(cv_scores)}
```

```
Average cross-validation score = 0.84
Standard deviation of cross-validation score = 0.09
```

Under the hood

- It creates `cv` folds on the data.
- In each fold, it fits the model on the training portion and scores on the validation portion.
- The output is a list of validation scores in each fold.

`cross_validate`

- Similar to `cross_val_score` but more powerful.
- Lets us access training and validation scores.

```
In [23]: 1 scores = cross_validate(model, X_train, y_train, cv=10, return_train_sc
2 pd.DataFrame(scores)
```

Out[23]:

	fit_time	score_time	test_score	train_score
0	0.002467	0.001434	0.764706	0.913333
1	0.001965	0.001553	0.823529	0.906667
2	0.001762	0.001211	0.705882	0.906667
3	0.001707	0.001211	0.941176	0.900000
4	0.002209	0.001237	0.823529	0.906667
5	0.001707	0.001201	0.823529	0.913333
6	0.001752	0.001191	0.705882	0.920000
7	0.001724	0.001166	0.937500	0.900662
8	0.001722	0.001112	0.937500	0.900662
9	0.001599	0.001074	0.937500	0.900662

```
In [24]: 1 pd.DataFrame(pd.DataFrame(scores).mean())
```

Out[24]:

	0
fit_time	0.001861
score_time	0.001239
test_score	0.840074
train_score	0.906865

Keep in mind that cross-validation does not return a model. It is not a way to build a model that can be applied to new data. The purpose of cross-validation is to **evaluate** how well the model will generalize to unseen data.

Note that both `cross_val_score` and `cross_validate` functions do not shuffle the data. Check out [StratifiedKFold](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html#sklearn.model_selection.StratifiedKFold) (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html#sklearn.model_selection.StratifiedKFold) where proportions of classes is the same in each fold as they are in the whole dataset. By default, `sklearn` uses `StratifiedKFold` when carrying out cross-validation for classification problems.

In []: 1 `mglearn.plots.plot_cross_validation()`

Our typical supervised learning set up is as follows:

- We are given training data with features `x` and target `y`
- We split the data into train and test portions: `x_train`, `y_train`, `x_test`, `y_test`
- We carry out hyperparameter optimization using cross-validation on the train portion: `x_train` and `y_train`.
- We assess our best performing model on the test portion: `x_test` and `y_test`.
- What we care about is the **test error**, which tells us how well our model can be generalized.
- If this test error is "reasonable" we deploy the model which will be used on new unseen examples.

In [27]: 1 `X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=`
2 `model = DecisionTreeClassifier(max_depth=10)`
3 `scores = cross_validate(model, X_train, y_train, cv=10, return_train_sc`
4 `pd.DataFrame(scores)`
5
6

Out[27]:

	fit_time	score_time	test_score	train_score
0	0.002738	0.001618	0.875000	1.000000
1	0.002283	0.001780	0.875000	0.992857
2	0.001879	0.001215	0.875000	1.000000
3	0.001745	0.001198	0.687500	1.000000
4	0.001719	0.001201	0.812500	1.000000
5	0.002033	0.001563	0.812500	1.000000
6	0.002237	0.001231	0.866667	0.985816
7	0.001750	0.001477	0.600000	0.992908
8	0.002210	0.001361	0.666667	1.000000
9	0.001799	0.001203	0.733333	1.000000

```
In [ ]: 1 def mean_std_cross_val_scores(model, X_train, y_train, **kwargs):
2     """
3         Returns mean and std of cross validation
4     """
5     scores = cross_validate(model, X_train, y_train, **kwargs)
6
7     mean_scores = pd.DataFrame(scores).mean()
8     std_scores = pd.DataFrame(scores).std()
9     out_col = []
10
11    for i in range(len(mean_scores)):
12        out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
13
14    return pd.Series(data=out_col, index=mean_scores.index)
```

```
In [ ]: 1 results = {}
2 results["Decision tree"] = mean_std_cross_val_scores(
3     model, X_train, y_train, return_train_score=True
4 )
5 pd.DataFrame(results).T
```

? ? Questions on cross-validation

1. k -fold cross-validation calls fit k times. True or False?
2. We use cross-validation to improve model performance. True or False?
3. Discuss advantages and disadvantages of cross-validation.

1. True
2. False. We can use it to assess model performance.

Break (5 min)



Underfitting, overfitting, the fundamental trade-off, the golden rule [[video \(<https://youtu.be/Ihay8yE5KTI>\)](https://youtu.be/Ihay8yE5KTI)]

Types of errors

Imagine that your train and validation errors do not align with each other. How do you diagnose the problem?

We're going to think about 4 types of errors:

- E_{train} is your training error (or mean train error from cross-validation).
- E_{valid} is your validation error (or mean validation error from cross-validation).
- E_{test} is your test error.
- E_{best} is the best possible error you could get for a given problem.

Underfitting

```
In [28]: 1 model = DecisionTreeClassifier(max_depth=1) # decision stump
2 scores = cross_validate(model, X_train, y_train, cv=10, return_train_score=True)
3 print("Train error: %0.3f" % (1 - np.mean(scores["train_score"])))
4 print("Validation error: %0.3f" % (1 - np.mean(scores["test_score"])))
```

Train error: 0.188
 Validation error: 0.212

- If your model is too simple, like `DummyClassifier` or `DecisionTreeClassifier` with `max_depth=1`, it's not going to pick up on some random quirks in the data but it won't even capture useful patterns in the training data.
- The model won't be very good in general. Both train and validation errors would be high. This is **underfitting**.
- The gap between train and validation error is going to be lower.
- $E_{\text{best}} < E_{\text{train}} \lesssim E_{\text{valid}}$

Overfitting

```
In [29]: 1 model = DecisionTreeClassifier(max_depth=None)
2 scores = cross_validate(model, X_train, y_train, cv=10, return_train_score=True)
3 print("Train error: %0.3f" % (1 - np.mean(scores["train_score"])))
4 print("Validation error: %0.3f" % (1 - np.mean(scores["test_score"])))
```

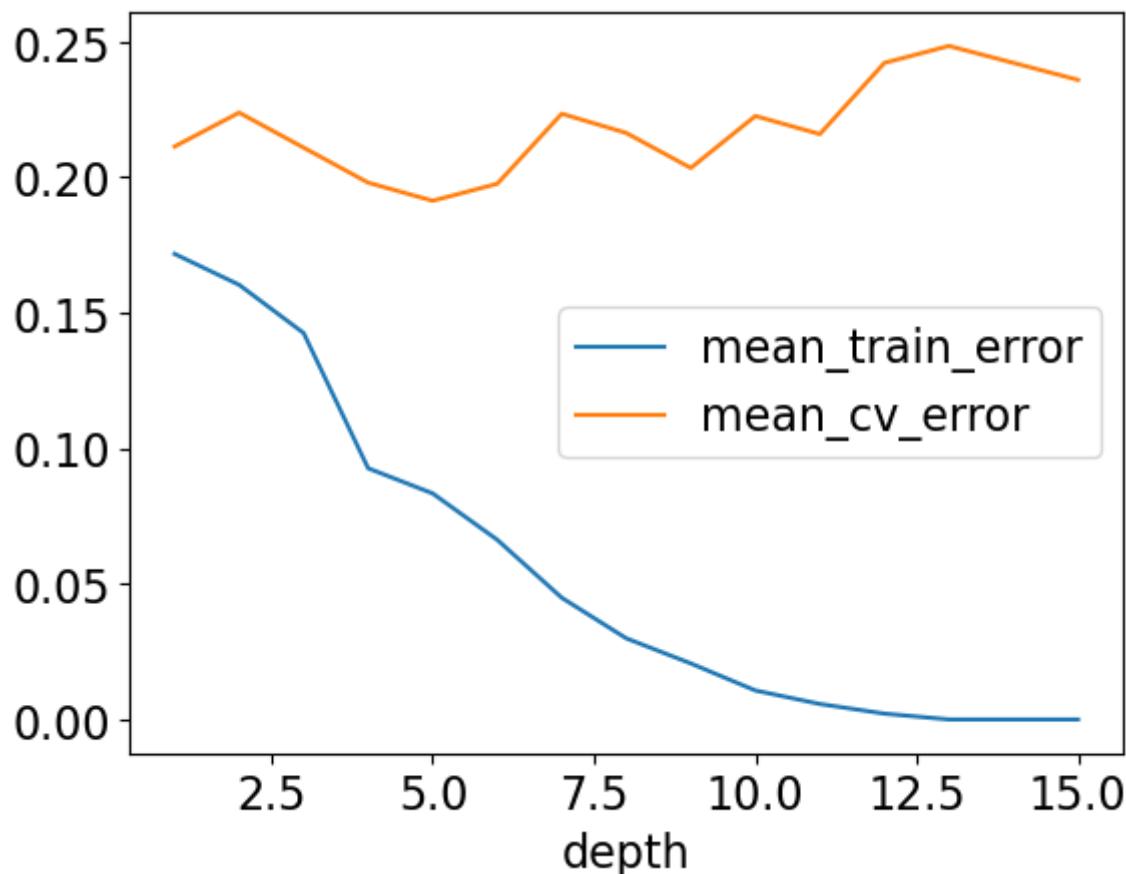
Train error: 0.000
 Validation error: 0.220

- If your model is very complex, like a `DecisionTreeClassifier(max_depth=None)`, then you will learn unreliable patterns in order to get every single training example correct.
- The training error is going to be very low but there will be a big gap between the training error and the validation error. This is **overfitting**.
- In overfitting scenario, usually we'll see: $E_{\text{train}} < E_{\text{best}} < E_{\text{valid}}$
- In general, if E_{train} is low, we are likely to be in the overfitting scenario. It is fairly common to have at least a bit of this.

- So the validation error does not necessarily decrease with the training error.

```
In [30]: 1 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
2 results_dict = {
3     "depth": [],
4     "mean_train_error": [],
5     "mean_cv_error": [],
6     "std_cv_error": [],
7     "std_train_error": [],
8 }
9 param_grid = {"max_depth": np.arange(1, 16)}
10
11 for depth in param_grid["max_depth"]:
12     model = DecisionTreeClassifier(max_depth=depth)
13     scores = cross_validate(model, X_train, y_train, cv=10, return_train_time=True)
14     results_dict["depth"].append(depth)
15     results_dict["mean_cv_error"].append(1 - np.mean(scores["test_score"]))
16     results_dict["mean_train_error"].append(1 - np.mean(scores["train_score"]))
17     results_dict["std_cv_error"].append(scores["test_score"].std())
18     results_dict["std_train_error"].append(scores["train_score"].std())
19
20 results_df = pd.DataFrame(results_dict)
21 results_df = results_df.set_index("depth")
```

```
In [31]: 1 results_df[["mean_train_error", "mean_cv_error"]].plot();
```



- Here, for larger depths we observe that the training error is close to 0 but validation error goes up and down.
- As we make more complex models we start encoding random quirks in the data, which are not grounded in reality.

- These random quirks do not generalize well to new data.
- This problem of failing to be able to generalize to the validation data or test data is called **overfitting**.

The "fundamental tradeoff" of supervised learning:

As you increase model complexity, E_{train} tends to go down but $E_{\text{valid}} - E_{\text{train}}$ tends to go up.

Bias vs variance tradeoff

- The fundamental trade-off is also called the bias/variance tradeoff in supervised machine learning.

Bias : the tendency to consistently learn the same wrong thing (high bias corresponds to underfitting), or failing to learn something important

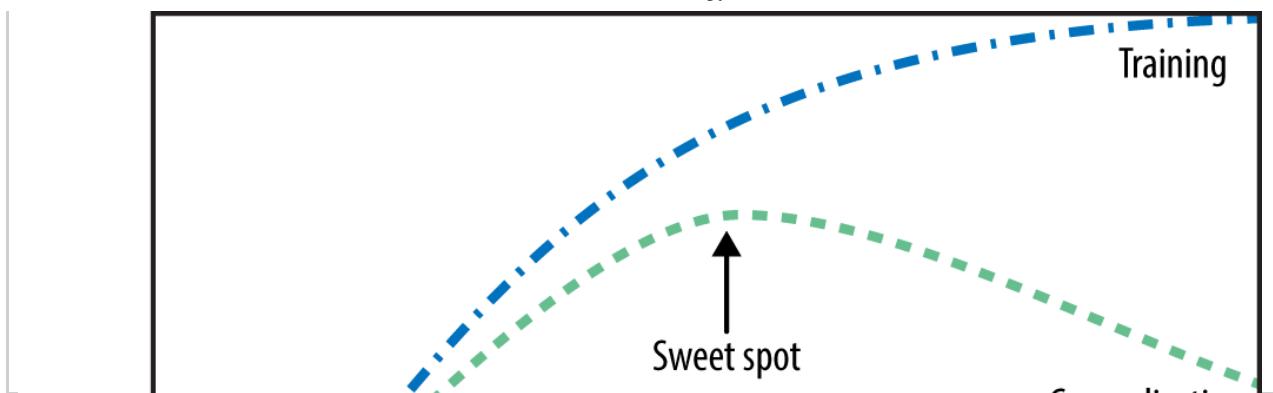
Variance : the tendency to learn random things irrespective of the real signal (high variance corresponds to overfitting)

Check out [this article by Pedro Domingos](#)

(<https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf>) for some approachable explanation on machine learning fundamentals and bias-variance tradeoff.

How to pick a model that would generalize better?

- We want to avoid both underfitting and overfitting.
- We want to be consistent with the training data but we don't rely too much on it.



- There are many subtleties here and there is no perfect answer but a common practice is to pick the model with minimum cross-validation error.

```
In [ ]: 1 def cross_validate_std(*args, **kwargs):
2     """Like cross_validate, except also gives the standard deviation of
3     res = pd.DataFrame(cross_validate(*args, **kwargs))
4     res_mean = res.mean()
5     res_mean["std_test_score"] = res["test_score"].std()
6     if "train_score" in res:
7         res_mean["std_train_score"] = res["train_score"].std()
8     return res_mean
```

```
In [32]: 1 results_df
```

```
Out[32]:    mean_train_error  mean_cv_error  std_cv_error  std_train_error
```

depth	mean_train_error	mean_cv_error	std_cv_error	std_train_error
1	0.171657	0.211250	0.048378	0.006805
2	0.160258	0.223750	0.062723	0.007316
3	0.142467	0.210833	0.067757	0.022848
4	0.092604	0.197917	0.056955	0.006531
5	0.083338	0.191250	0.067120	0.010650
6	0.066251	0.197500	0.074773	0.012019
7	0.044873	0.223333	0.080734	0.009059
8	0.029909	0.216250	0.088397	0.009422
9	0.020653	0.203333	0.090978	0.010294
10	0.010679	0.222500	0.092669	0.007938
11	0.005699	0.215833	0.109335	0.004264
12	0.002143	0.242083	0.095497	0.003273
13	0.000000	0.248333	0.089485	0.000000
14	0.000000	0.242083	0.086932	0.000000
15	0.000000	0.235833	0.083836	0.000000

test score vs. cross-validation score

```
In [33]: 1 best_depth = results_df.index.values[np.argmin(results_df["mean_cv_error"])
2 print(
3     "The minimum validation error is %0.3f at max_depth = %d "
4     %
5         np.min(results_df["mean_cv_error"]),
6         best_depth,
7     )
8 )
```

The minimum validation error is 0.191 at max_depth = 5

- Let's pick `max_depth = 5` and try this model on the test set.

```
In [34]: 1 model = DecisionTreeClassifier(max_depth=best_depth)
2 model.fit(X_train, y_train)
3 print(f"Error on test set: {1 - model.score(X_test, y_test):.2f}")
```

Error on test set: 0.19

- The test error is comparable with the cross-validation error.
- Do we feel confident that this model would give similar performance when deployed?

The golden rule

- Even though we care the most about test error **THE TEST DATA CANNOT INFLUENCE THE TRAINING PHASE IN ANY WAY.**
- We have to be very careful not to violate it while developing our ML pipeline.
- Even experts end up breaking it sometimes which leads to misleading results and lack of generalization on the real data.

Golden rule violation: Example 1

Emergent Tech ▶ Artificial Intelligence

Was this quake AI a little too artificial? Nature-published research accused of boosting accuracy by mixing training, testing data

Academics, journal deny making a boo boo

By [Katyanna Quach](#) 3 Jul 2019 at 09:01

21 SHARE ▾



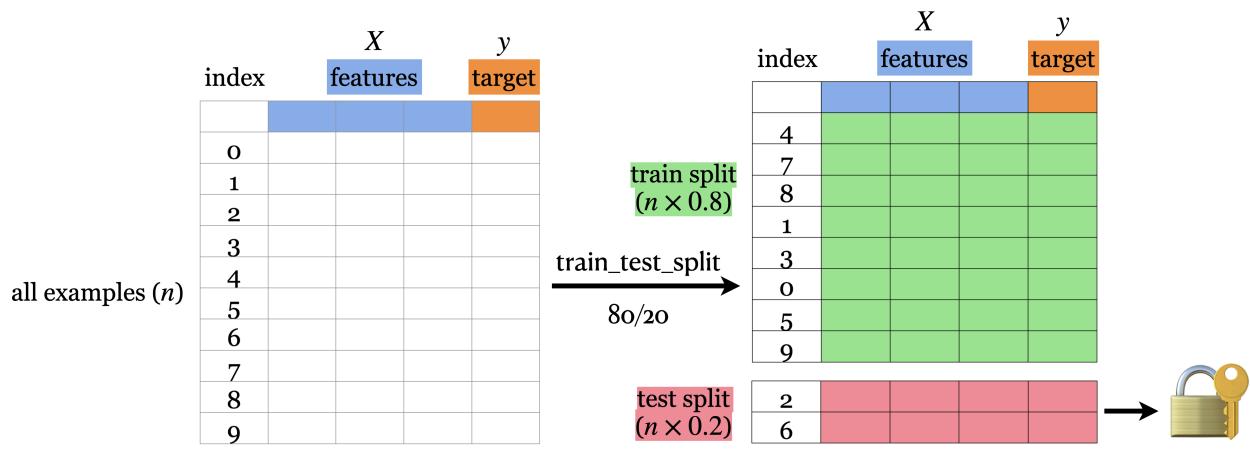
Golden rule violation: Example 2



Intelligent Machines

How can we avoid violating golden rule?

- Recall that when we split data, we put our test set in an imaginary vault.



Here is the workflow we'll generally follow.

- Splitting:** Before doing anything, split the data x and y into x_{train} , x_{test} , y_{train} , y_{test} or train_df and test_df using `train_test_split`.
- Select the best model using cross-validation:** Use `cross_validate` with `return_train_score = True` so that we can get access to training scores in each fold. (If we want to plot train vs validation error plots, for instance.)
- Scoring on test data:** Finally score on the test data with the chosen hyperparameters to examine the generalization performance.

Again, there are many subtleties here we'll discuss the golden rule multiple times throughout the course and in the program.

?? Questions for you

Underfitting or overfitting?

1. If the mean train accuracy is much higher than the mean cross-validation accuracy.
2. If the mean train accuracy and the mean cross-validation accuracy are both low and relatively similar in value.
3. Decision tree with no limit on the depth.
4. Decision stump on a complicated classification problem.

1. Overfitting
2. Underfitting
3. Overfitting
4. Underfitting

State whether True/False.

1. In supervised learning, the training error is always lower than the validation error.
2. The fundamental tradeoff of ML states that as training error goes down, validation error goes up.
3. More "complicated" models are more likely to overfit than "simple" ones.
4. If our training error is extremely low, we are likely to be overfitting.
5. A very simple model (e.g. decision stump) has high variance.

1. False
2. False
3. True
4. True
5. False

What did we learn today?

- Importance of generalization in supervised machine learning
- Data splitting as a way to approximate generalization error
- Train, test, validation, deployment data
- Cross-validation
- A typical sequence of steps to train supervised machine learning models
 - training the model on the train split
 - tuning hyperparameters using the validation split
 - checking the generalization performance on the test split

- Overfitting, underfitting, the fundamental tradeoff, and the golden rule.



In []:

1

CPSC 330

Applied Machine Learning

Lecture 4: *k*-Nearest Neighbours and SVM RBFs

UBC 2022-23

Instructor: Mathias Lécuyer

If two things are similar, the thought of one will tend to trigger the thought of the other
-- Aristotle

Imports

In [1]:

```
1 import sys
2
3 import IPython
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import pandas as pd
7 from IPython.display import HTML
8
9 sys.path.append("../code/.")
10
11 import ipywidgets as widgets
12 import mglearn
13 from IPython.display import display
14 from ipywidgets import interact, interactive
15 from plotting_functions import *
16 from sklearn.dummy import DummyClassifier
17 from sklearn.model_selection import cross_validate, train_test_split
18 from utils import *
19
20 %matplotlib inline
21
22 pd.set_option("display.max_colwidth", 200)
23 import warnings
24
25 warnings.filterwarnings("ignore")
```

Learning outcomes

From this lecture, you will be able to

- explain the notion of similarity-based algorithms;
- broadly describe how k -NNs use distances;
- discuss the effect of using a small/large value of the hyperparameter k when using the k -NN algorithm;
- describe the problem of curse of dimensionality;
- explain the general idea of SVMs with RBF kernel;
- broadly describe the relation of `gamma` and `c` hyperparameters of SVMs with the fundamental tradeoff.

If you want to run this notebook you will have to install `ipywidgets`. Follow the installation instructions [here](https://ipywidgets.readthedocs.io/en/latest/user_install.html) (https://ipywidgets.readthedocs.io/en/latest/user_install.html).

Motivation and distances [[video](#) (<https://youtu.be/hCa3EXEUmQk>)]

Analogy-based models

- Suppose you are given the following training examples with corresponding labels and are asked to label a given test example.

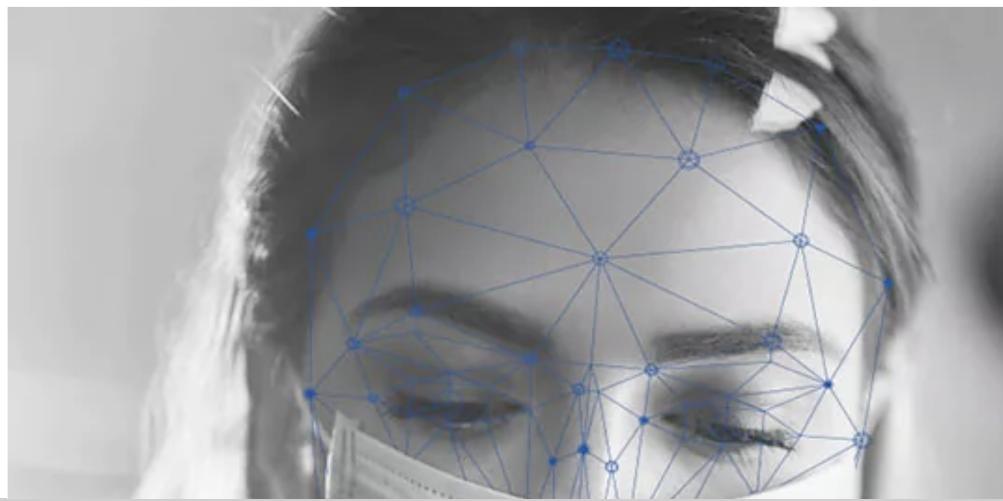


[source \(https://vipl.ict.ac.cn/en/database.php\)](https://vipl.ict.ac.cn/en/database.php)

- An intuitive way to classify the test example is by finding the most "similar" example(s) from the training set and using that label for the test example.

Analogy-based algorithms in practice

- [Herta's High-tech Facial Recognition \(<https://www.hertasecurity.com/en>\)](https://www.hertasecurity.com/en)
 - Feature vectors for human faces
 - \$k\$-NN to identify which face is on their watch list
- Recommendation systems

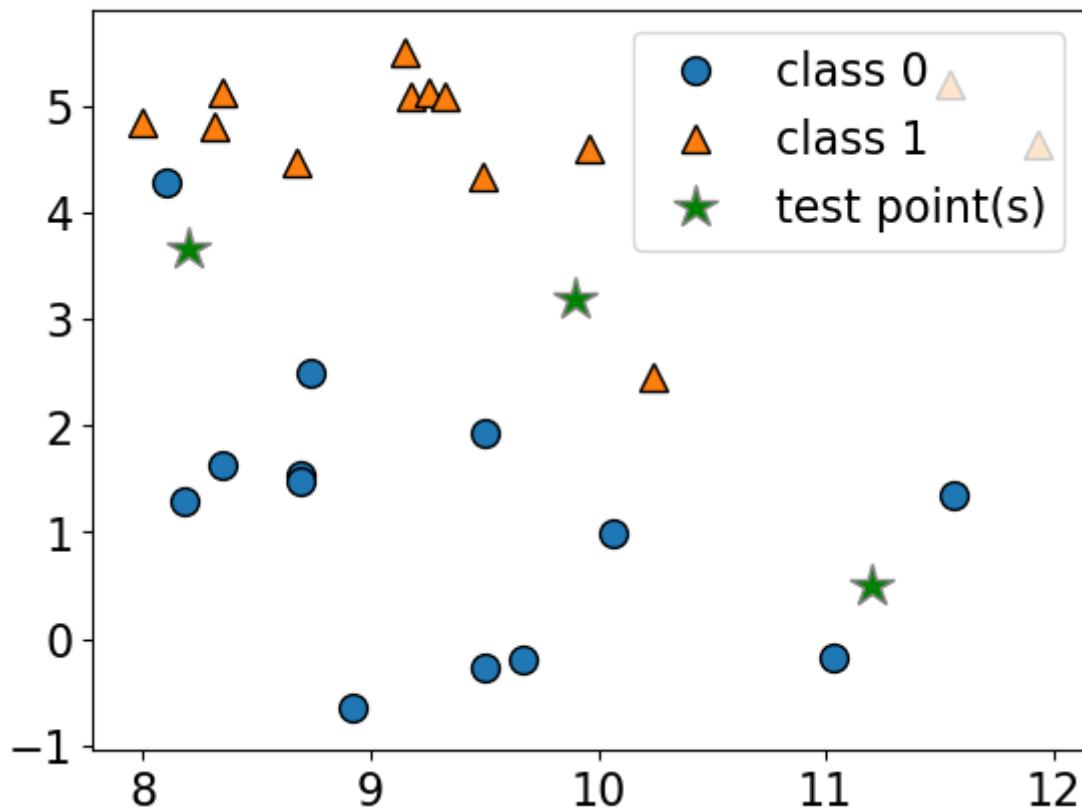


General idea of k -nearest neighbours algorithm

- Consider the following toy dataset with two classes.
 - blue circles \rightarrow class 0
 - red triangles \rightarrow class 1
 - green stars \rightarrow test examples

```
In [2]: 1 X, y = mglearn.datasets.make_forge()
2 X_test = np.array([[8.2, 3.66214339], [9.9, 3.2], [11.2, 0.5]])
```

```
In [3]: 1 plot_train_test_points(X, y, X_test)
```

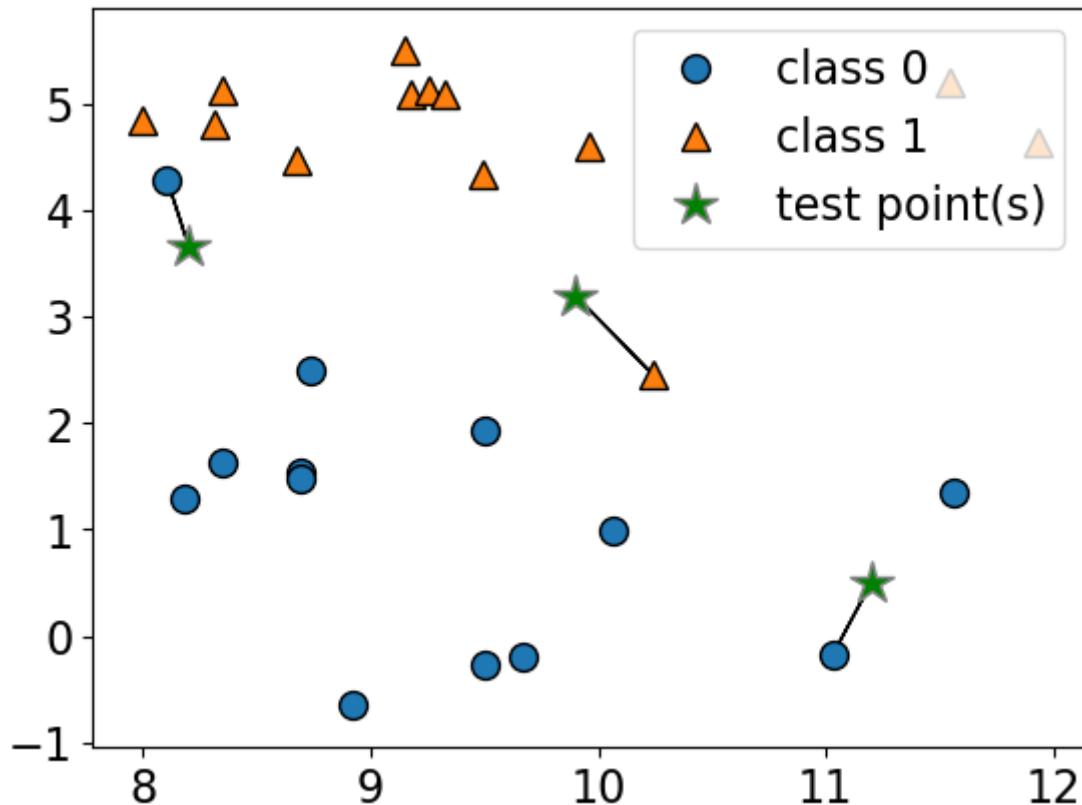


- Given a new data point, predict the class of the data point by finding the "closest" data point in the training set, i.e., by finding its "nearest neighbour" or majority vote of nearest neighbours.

```
In [4]: 1 def f(n_neighbors):
2     return plot_knn_clf(X, y, X_test, n_neighbors=n_neighbors)
```

```
In [5]: 1 interactive(
2     f,
3     n_neighbors=widgets.IntSlider(min=1, max=7, step=2, value=1),
4 )
```

n_neighbors 1



```
Out[5]: interactive(children=(IntSlider(value=1, description='n_neighbors', max=7, min=1, step=2), Output()), _dom_cla...
```

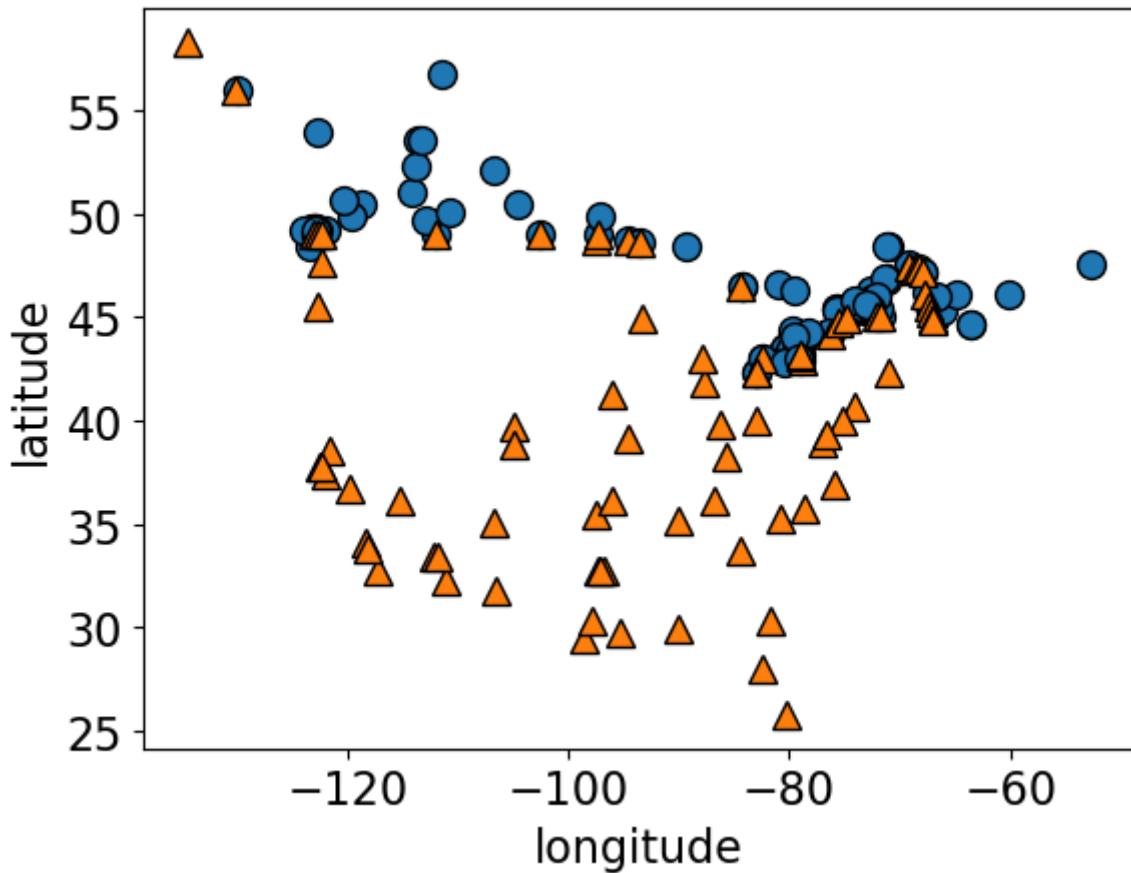
Geometric view of tabular data and dimensions

- To understand analogy-based algorithms it's useful to think of data as points in a high dimensional space.
- Our x represents the problem in terms of relevant **features** (d) with one dimension for each **feature** (column).
- Examples are **points in a d -dimensional space**.

How many dimensions (features) are there in the cities data?

```
In [7]: 1 cities_df = pd.read_csv("../data/canada_usa_cities.csv")
2 X_cities = cities_df[["longitude", "latitude"]]
3 y_cities = cities_df["country"]
```

```
In [8]: 1 mglearn.discrete_scatter(X_cities.iloc[:, 0], X_cities.iloc[:, 1], y_ci
2 plt.xlabel("longitude")
3 plt.ylabel("latitude");
```



- Recall the [Spotify Song Attributes](#) (<https://www.kaggle.com/geomack/spotifyclassification/home>) dataset from homework 1.
- How many dimensions (features) we used in the homework?

```
In [9]: 1 spotify_df = pd.read_csv("../data/spotify.csv", index_col=0)
2 X_spotify = spotify_df.drop(columns=["target", "song_title", "artist"])
3 print("The number of features in the Spotify dataset: %d" % X_spotify.s
4 X_spotify.head()
```

The number of features in the Spotify dataset: 13

Out[9]:

	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode
0	0.0102	0.833	204600	0.434	0.021900	2	0.1650	-8.795	1
1	0.1990	0.743	326933	0.359	0.006110	1	0.1370	-10.401	1
2	0.0344	0.838	185707	0.412	0.000234	2	0.1590	-7.148	1
3	0.6040	0.494	199413	0.338	0.510000	5	0.0922	-15.236	1
4	0.1800	0.678	392893	0.561	0.512000	5	0.4390	-11.648	0

Dimensions in ML problems

In ML, usually we deal with high dimensional problems where examples are hard to visualize.

- $d \approx 20$ is considered low dimensional
- $d \approx 1000$ is considered medium dimensional
- $d \approx 100,000$ is considered high dimensional

Feature vectors

Feature vector : is composed of feature values associated with an example.

Some example feature vectors are shown below.

```
In [10]: 1 print(
2     "An example feature vector from the cities dataset: %s"
3     % (X_cities.iloc[0].to_numpy())
4 )
5 print(
6     "An example feature vector from the Spotify dataset: \n%s"
7     % (X_spotify.iloc[0].to_numpy())
8 )
```

An example feature vector from the cities dataset: [-130.0437 55.9773]
An example feature vector from the Spotify dataset:
[1.02000e-02 8.33000e-01 2.04600e+05 4.34000e-01 2.19000e-02
 2.00000e+00 1.65000e-01 -8.79500e+00 1.00000e+00 4.31000e-01
 1.50062e+02 4.00000e+00 2.86000e-01]

Similarity between examples

Let's take 2 points (two feature vectors) from the cities dataset.

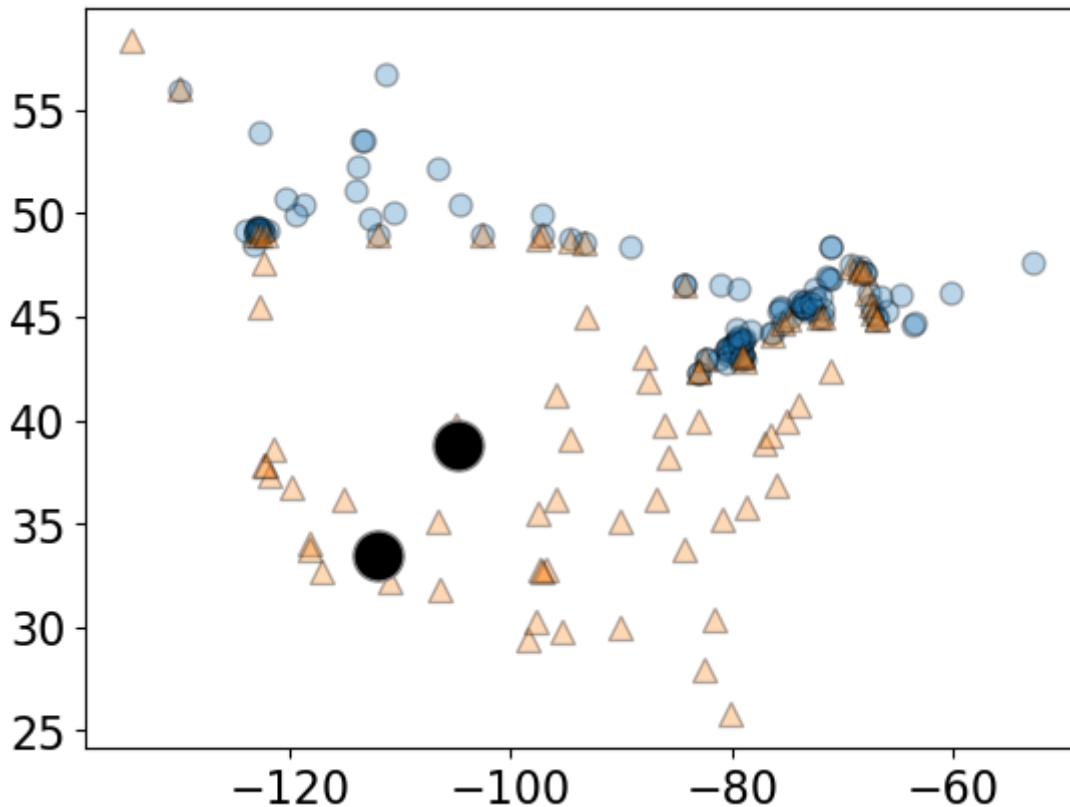
```
In [11]: 1 two_cities = X_cities.sample(2, random_state=120)
2 two_cities
```

Out[11]: longitude latitude

	longitude	latitude
69	-104.8253	38.8340
35	-112.0741	33.4484

The two sampled points are shown as big black circles.

```
In [12]: 1 mglearn.discrete_scatter(
2     X_cities.iloc[:, 0], X_cities.iloc[:, 1], y_cities, s=8, alpha=0.3
3 )
4 mglearn.discrete_scatter(
5     two_cities.iloc[:, 0], two_cities.iloc[:, 1], markers="o", c="k", s
6 );
```



Distance between feature vectors

- For the cities at the two big circles, what is the *distance* between them?
- A common way to calculate the distance between vectors is calculating the **Euclidean distance**.
- The euclidean distance between vectors $u = \langle u_1, u_2, \dots, u_n \rangle$ and $v = \langle v_1, v_2, \dots, v_n \rangle$ is defined as:

$$\text{distance}(u, v) = \sqrt{\sum_{i=1}^n (u_i - v_i)^2}$$

Euclidean distance

In [13]: 1 two_cities

Out[13]:

	longitude	latitude
69	-104.8253	38.8340
35	-112.0741	33.4484

- Subtract the two cities
- Square the difference
- Sum them up
- Take the square root

In [14]:

```

1 # Subtract the two cities
2 print("Subtract the cities: \n%s\n" % (two_cities.iloc[1] - two_cities.
3
4 # Squared sum of the difference
5 print(
6     "Sum of squares: %0.4f" % (np.sum((two_cities.iloc[1] - two_cities.
7 )
8
9 # Take the square root
10 print(
11     "Euclidean distance between cities: %0.4f"
12     % (np.sqrt(np.sum((two_cities.iloc[1] - two_cities.iloc[0]) ** 2)))
13 )

```

Subtract the cities:
longitude -7.2488
latitude -5.3856
dtype: float64

Sum of squares: 81.5498
Euclidean distance between cities: 9.0305

In [15]: 1 two_cities

Out[15]:

	longitude	latitude
69	-104.8253	38.8340
35	-112.0741	33.4484

In [16]:

```

1 # Euclidean distance using sklearn
2 from sklearn.metrics.pairwise import euclidean_distances
3
4 euclidean_distances(two_cities)

```

Out[16]: array([[0. , 9.03049217],
 [9.03049217, 0.]])

Note: scikit-learn supports a number of other [distance metrics \(https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html\)](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html).

Finding the nearest neighbour

- Let's look at distances from all cities to all other cities

```
In [17]: 1 dists = euclidean_distances(X_cities)
2 np.fill_diagonal(dists, np.inf)
3 print("All distances: %s\n\n%s" % (dists.shape, dists))
```

All distances: (209, 209)

```
[[      inf  4.95511263  9.869531    ... 52.42640992 58.03345923
  51.49856241]
 [ 4.95511263           inf 14.6775792   ... 57.25372435 62.77196948
  56.25216034]
 [ 9.869531   14.6775792           inf   ... 44.23515175 50.24972011
  43.69922405]
 ...
 [52.42640992 57.25372435 44.23515175 ...           inf   6.83784786
  3.32275537]
 [58.03345923 62.77196948 50.24972011 ...   6.83784786           inf
  6.55573969]
 [51.49856241 56.25216034 43.69922405 ...   3.32275537 6.55573969
  inf]]
```

Let's look at the distances between City 0 and some other cities.

```
In [18]: 1 print("Feature vector for city 0: \n%s\n" % (X_cities.iloc[0]))
2 print("Distances from city 0 to the first 5 cities: %s" % (dists[0][:5])
3 # We can find the closest city with `np.argmin`:
4 print(
5     "The closest city from city 0 is: %d \nwith feature vector: \n%s"
6     % (np.argmin(dists[0]), X_cities.iloc[np.argmin(dists[0])]))
7 )
```

Feature vector for city 0:
longitude -130.0437
latitude 55.9773
Name: 0, dtype: float64

Distances from city 0 to the first 5 cities: [inf 4.95511263 9.
 869531 10.10645223 10.44966612]
The closest city from city 0 is: 81

with feature vector:
longitude -129.9912
latitude 55.9383
Name: 81, dtype: float64

Ok, so the closest city to City 0 is City 81.

Question

- Why did we set the diagonal entries to infinity before finding the closest city?

Finding the distances to a query point

We can also find the distances to a new "test" or "query" city:

```
In [19]: 1 # Let's find a city that's closest to the a query city
2 query_point = [[-80, 25]]
3
4 dists = euclidean_distances(X_cities, query_point)
5 dists[0:10]
```

```
Out[19]: array([[58.85545875],
 [63.80062924],
 [49.30530902],
 [49.01473536],
 [48.60495488],
 [39.96834506],
 [32.92852376],
 [29.53520104],
 [29.52881619],
 [27.84679073]])
```

```
In [20]: 1 # The query point is closest to
2 print(
3     "The query point %s is closest to the city with index %d and the di
4     % (query_point, np.argmin(dists), dists[np.argmin(dists)])
5 )
```

The query point `[-80, 25]` is closest to the city with index 72 and the distance between them is: 0.7982

\$k\$-Nearest Neighbours (\$k\$-NNs) [video](<https://youtu.be/bENDqXKJLmg>)

In [21]:

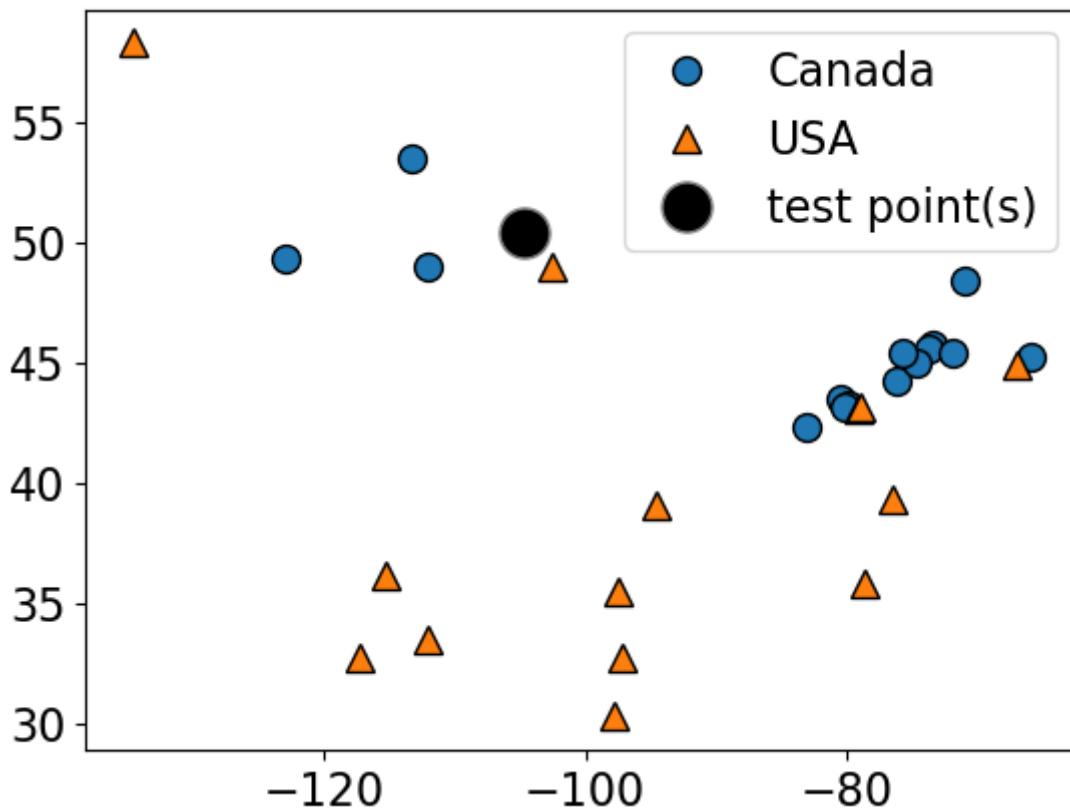
```
1 small_cities = cities_df.sample(30, random_state=90)
2 one_city = small_cities.sample(1, random_state=44)
3 small_train_df = pd.concat([small_cities, one_city]).drop_duplicates(ke
```

In [22]:

```
1 X_small_cities = small_train_df.drop(columns=["country"]).to_numpy()
2 y_small_cities = small_train_df["country"].to_numpy()
3 test_point = one_city[["longitude", "latitude"]].to_numpy()
```

In [23]:

```
1 plot_train_test_points(
2     X_small_cities,
3     y_small_cities,
4     test_point,
5     class_names=["Canada", "USA"],
6     test_format="circle",
7 )
```



- Given a new data point, predict the class of the data point by finding the "closest" data point in the training set, i.e., by finding its "nearest neighbour" or majority vote of nearest neighbours.

Suppose we want to predict the class of the black point.

- An intuitive way to do this is predict the same label as the "closest" point ($k = 1$) (1-nearest neighbour)
- We would predict a target of **USA** in this case.

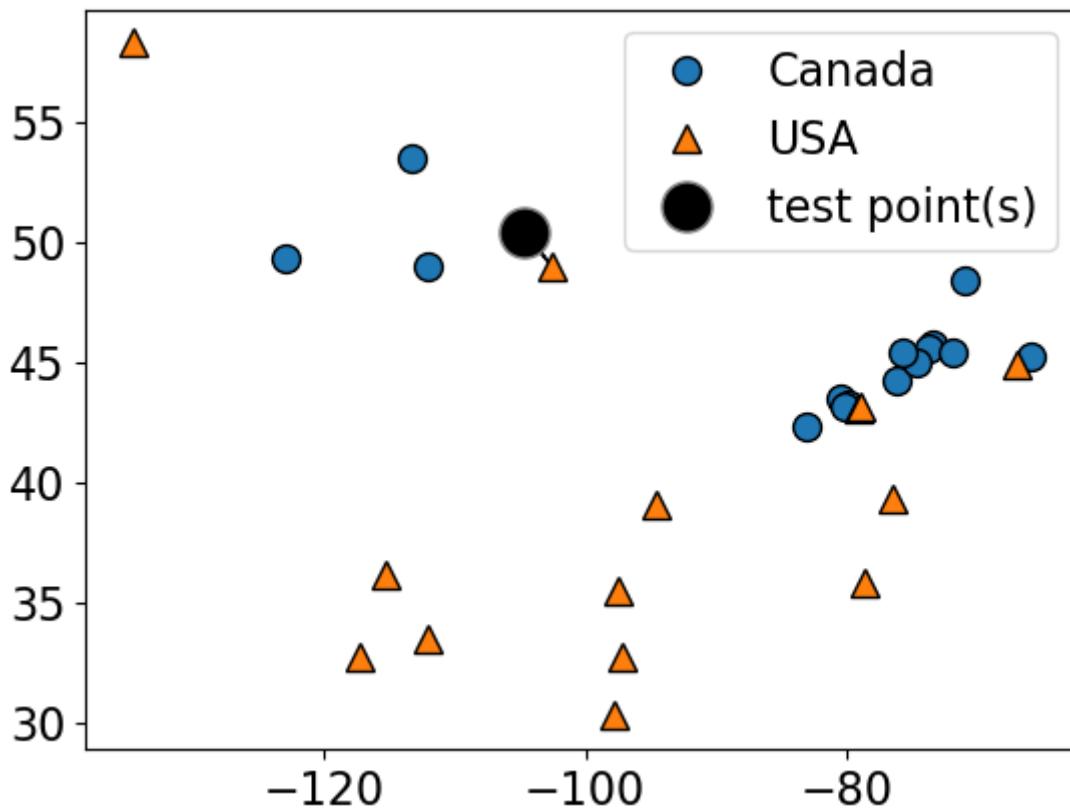
In [24]:

```

1 plot_knn_clf(
2     x_small_cities,
3     y_small_cities,
4     test_point,
5     n_neighbors=1,
6     class_names=["Canada", "USA"],
7     test_format="circle",
8 )

```

n_neighbors 1



How about using $k > 1$ to get a more robust estimate?

- For example, we could also use the 3 closest points ($k = 3$) and let them **vote** on the correct class.
- The **Canada** class would win in this case.

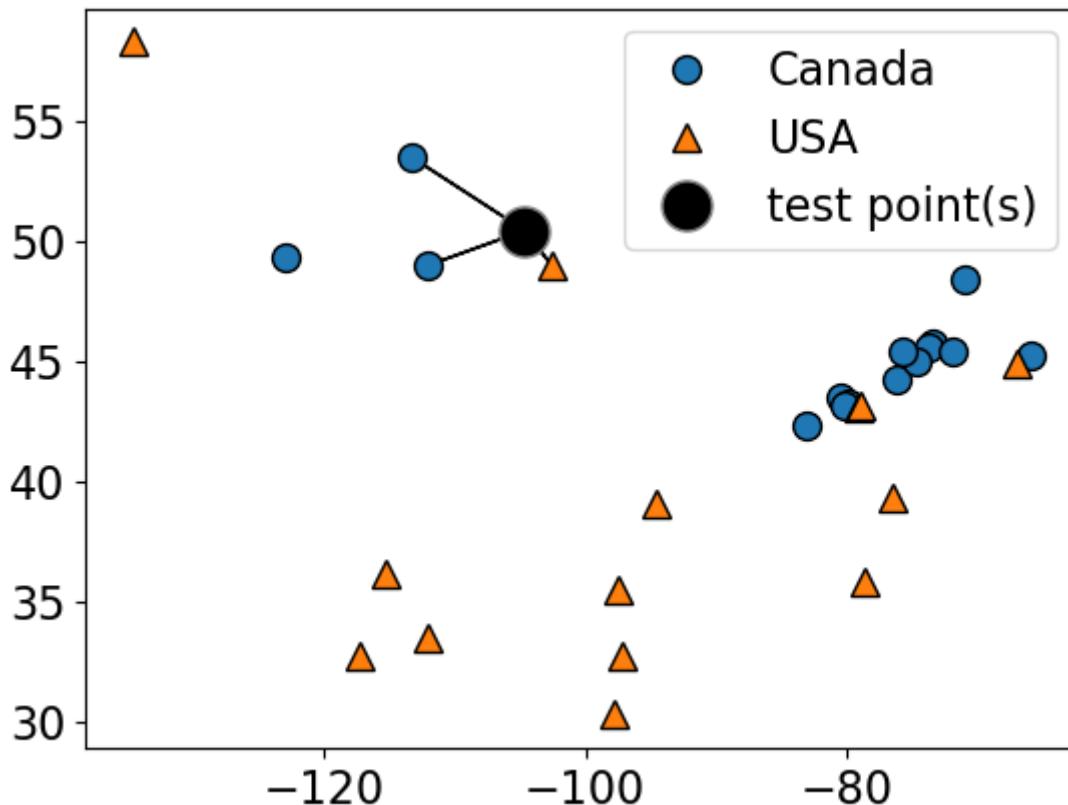
In [25]:

```

1 plot_knn_clf(
2     X_small_cities,
3     y_small_cities,
4     test_point,
5     n_neighbors=3,
6     class_names=["Canada", "USA"],
7     test_format="circle",
8 )

```

n_neighbors 3



In [26]:

```

1 from sklearn.neighbors import KNeighborsClassifier
2
3 k_values = [1, 3]
4
5 for k in k_values:
6     neigh = KNeighborsClassifier(n_neighbors=k)
7     neigh.fit(X_small_cities, y_small_cities)
8     print(
9         "Prediction of the black dot with %d neighbours: %s"
10        % (k, neigh.predict(test_point)))
11

```

Prediction of the black dot with 1 neighbours: ['USA']
 Prediction of the black dot with 3 neighbours: ['Canada']

Question

- Is it a good or a bad idea to consider an odd number for k ? Why or why not?

Choosing `n_neighbors`

- The primary hyperparameter of the model is `n_neighbors` (k) which decides how many neighbours should vote during prediction?
- What happens when we play around with `n_neighbors` ?
- Are we more likely to overfit with a low `n_neighbors` or a high `n_neighbors` ?
- Let's examine the effect of the hyperparameter on our cities data.

In [27]:

```

1 X = cities_df.drop(columns=[ "country" ])
2 y = cities_df[ "country" ]
3
4 # split into train and test sets
5 X_train, X_test, y_train, y_test = train_test_split(
6     X, y, test_size=0.1, random_state=123
7 )

```

In [28]:

```

1 k = 1
2 knn1 = KNeighborsClassifier(n_neighbors=k)
3 scores = cross_validate(knn1, X_train, y_train, return_train_score=True)
4 pd.DataFrame(scores)

```

Out[28]:

	fit_time	score_time	test_score	train_score
0	0.003854	0.006703	0.710526	1.0
1	0.001873	0.002923	0.684211	1.0
2	0.001859	0.002859	0.842105	1.0
3	0.001891	0.003029	0.702703	1.0
4	0.001831	0.002857	0.837838	1.0

In [29]:

```

1 k = 100
2 knn100 = KNeighborsClassifier(n_neighbors=k)
3 scores = cross_validate(knn100, X_train, y_train, return_train_score=True)
4 pd.DataFrame(scores)

```

Out[29]:

	fit_time	score_time	test_score	train_score
0	0.002043	0.141483	0.605263	0.600000
1	0.001216	0.002506	0.605263	0.600000
2	0.001294	0.002617	0.605263	0.600000
3	0.001195	0.002623	0.594595	0.602649
4	0.001272	0.002432	0.594595	0.602649

```
In [30]: 1 def f(n_neighbors=1):
2     results = {}
3     knn = KNeighborsClassifier(n_neighbors=n_neighbors)
4     scores = cross_validate(knn, X_train, y_train, return_train_score=True)
5     results["n_neighbours"] = [n_neighbors]
6     results["mean_train_score"] = [round(scores["train_score"].mean(), 3)]
7     results["mean_valid_score"] = [round(scores["test_score"].mean(), 3)]
8     print(pd.DataFrame(results))
9
10
11 interactive(
12     f,
13     n_neighbors=widgets.IntSlider(min=1, max=101, step=10, value=1),
14 )
```

Out[30]: `interactive(children=(IntSlider(value=1, description='n_neighbors', max=101, min=1, step=10), Output()), _dom_...`

```
In [1]: 1 plot_knn_decision_boundaries(X_train, y_train, k_values=[1, 11, 100])
```

```
-----
-- NameError                                                 Traceback (most recent call last)
t)
Cell In[1], line 1
----> 1 plot_knn_decision_boundaries(X_train, y_train, k_values=[1, 11, 100])

NameError: name 'plot_knn_decision_boundaries' is not defined
```

How to choose `n_neighbors` ?

- `n_neighbors` is a hyperparameter
- We can use hyperparameter optimization to choose `n_neighbors` .

In [32]:

```

1 results_dict = {
2     "n_neighbors": [],
3     "mean_train_score": [],
4     "mean_cv_score": [],
5     "std_cv_score": [],
6     "std_train_score": [],
7 }
8 param_grid = {"n_neighbors": np.arange(1, 50, 5)}
9
10 for k in param_grid["n_neighbors"]:
11     knn = KNeighborsClassifier(n_neighbors=k)
12     scores = cross_validate(knn, X_train, y_train, return_train_score=True)
13     results_dict["n_neighbors"].append(k)
14
15     results_dict["mean_cv_score"].append(np.mean(scores["test_score"]))
16     results_dict["mean_train_score"].append(np.mean(scores["train_score"]))
17     results_dict["std_cv_score"].append(scores["test_score"].std())
18     results_dict["std_train_score"].append(scores["train_score"].std())
19
20 results_df = pd.DataFrame(results_dict)

```

In [33]:

```

1 results_df = results_df.set_index("n_neighbors")
2 results_df

```

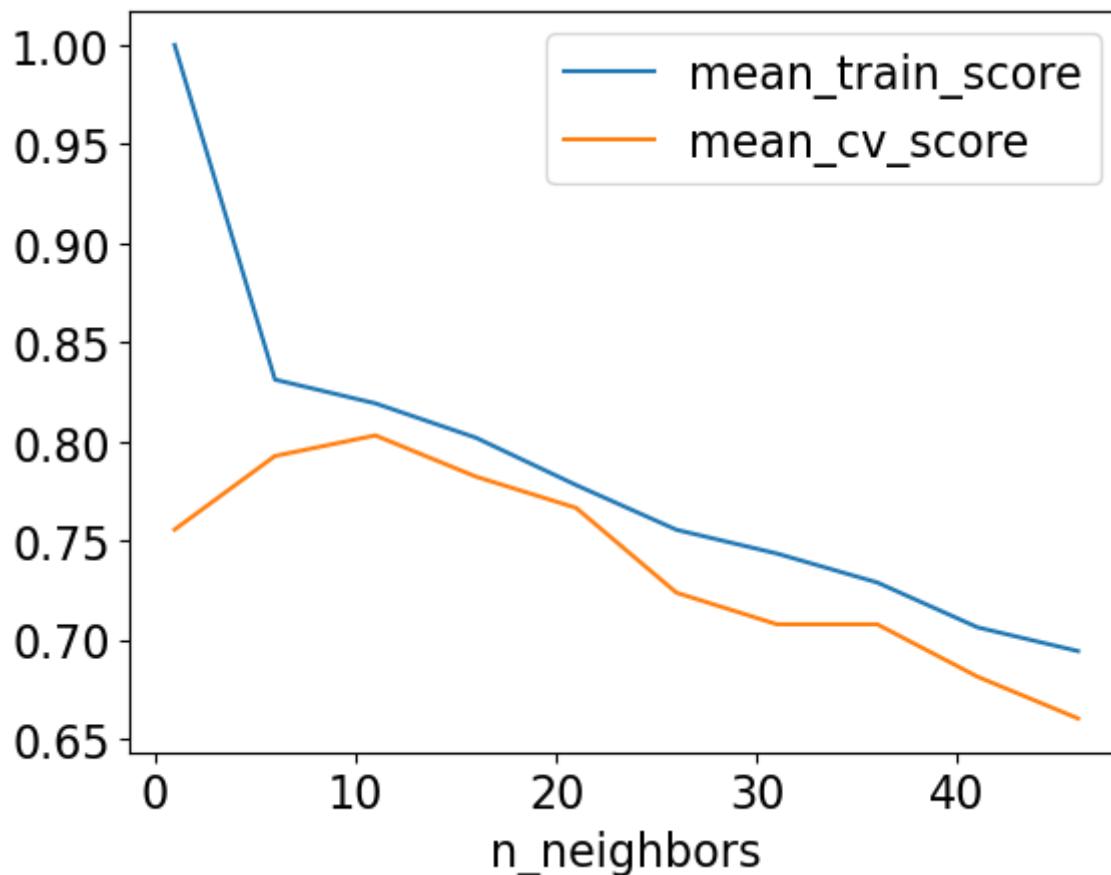
Out[33]:

	mean_train_score	mean_cv_score	std_cv_score	std_train_score
--	------------------	---------------	--------------	-----------------

n_neighbors	mean_train_score	mean_cv_score	std_cv_score	std_train_score
1	1.000000	0.755477	0.069530	0.000000
6	0.831135	0.792603	0.046020	0.013433
11	0.819152	0.802987	0.041129	0.011336
16	0.801863	0.782219	0.074141	0.008735
21	0.777934	0.766430	0.062792	0.016944
26	0.755364	0.723613	0.061937	0.025910
31	0.743391	0.707681	0.057646	0.030408
36	0.728777	0.707681	0.064452	0.021305
41	0.706128	0.681223	0.061241	0.018310
46	0.694155	0.660171	0.093390	0.018178

```
In [34]: 1 results_df[["mean_train_score", "mean_cv_score"]].plot()
```

```
Out[34]: <AxesSubplot: xlabel='n_neighbors'>
```



```
In [35]: 1 best_n_neighbours = results_df.idxmax()["mean_cv_score"]
2 best_n_neighbours
```

```
Out[35]: 11
```

Let's try our best model on test data.

```
In [36]: 1 knn = KNeighborsClassifier(n_neighbors=best_n_neighbours)
2 knn.fit(X_train, y_train)
3 print("Test accuracy: %0.3f" % (knn.score(X_test, y_test)))
```

Test accuracy: 0.905

1. Analogy-based models find examples from the test set that are most similar to the query example we are predicting.
2. A dataset with 10 dimensions is considered low dimensional.
3. Euclidean distance will always have a positive value.

? ? Questions on distances and \$k\\$-NNs

Exercise 4.1

What would be the Euclidean distance between the following two vectors u and v ?

```
In [41]: 1 u = np.array([0, 0, 20, -2])
2 v = np.array([-1, 0, 18, -4])
3
4 np.sum((u-v)*(u-v))
```

Out[41]: 9

Exercise 4.2 \$k\$-NN practice questions

- When we calculated Euclidean distances from all cities to all other cities, why did we set the diagonal entries to infinity before finding the closest city?
- Consider this toy dataset:

$\begin{aligned} X &= \begin{bmatrix} 5 & 2 & 4 & 3 & 2 & 2 & 10 & 10 & 9 & -1 & 9 \\ 0 & 0 & 1 & 1 & 1 & 2 \end{bmatrix}, \quad y = \\ &\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 2 \end{bmatrix}. \end{aligned}$

- If $k=1$, what would you predict for $x=\begin{bmatrix} 0 & 0 \end{bmatrix}$?
- If $k=3$, what would you predict for $x=\begin{bmatrix} 0 & 0 \end{bmatrix}$?

Recap - \$k\$-NNs

At home, you watched videos on analogy-based models, in particular k -nearest neighbors.

Let's answer some questions about what you have learned (T/F):

- Unlike with decision trees, with k -NNs most of the work is done at the `predict` stage.
- With k -NN, setting the hyperparameter k to larger values typically reduces training error.
- Similar to decision trees, k -NNs finds a small set of good features.
- In k -NN, the classification of the closest neighbour to the test example always contributes the most to the prediction.

Open questions

- Is it a good or a bad idea to consider an odd number for k ? Why or why not?
- In the video, diagonal entries of the Euclidian distance matrix were changed from 0 to infinity. Why?
- Why is the training error always 0.0 when $k = 1$?

More on k -NNs [[video](https://youtu.be/lRGbqi5S9gQ) (<https://youtu.be/lRGbqi5S9gQ>)]

Other useful arguments of `KNeighborsClassifier`

- `weights` \rightarrow When predicting label, you can assign higher weight to the examples which are closer to the query example.
- Exercise for you: Play around with this argument. Do you get a better validation score?

(Optional) Regression with k -nearest neighbours (k -NNs)

- Can we solve regression problems with k -nearest neighbours algorithm?
- In k -NN regression we take the average of the k -nearest neighbours.
- We can also have weighted regression.

See an example of regression in the lecture notes.

```
In [ ]: 1 mlearn.plots.plot_knn_regression(n_neighbors=1)
```

```
In [ ]: 1 mlearn.plots.plot_knn_regression(n_neighbors=3)
```

Pros of k -NNs for supervised learning

- Easy to understand, interpret.
- Simple hyperparameter k (`n_neighbors`) controlling the fundamental tradeoff.
- Can learn very complex functions given enough data.
- Lazy learning: Takes no time to `fit`

Cons of k -NNs for supervised learning

- Can be potentially be VERY slow during prediction time, especially when the training set is very large.
- Often not that great test accuracy compared to the modern approaches.
- It does not work well on datasets with many features or where most feature values are 0 most of the time (sparse datasets).

For regular k -NN for supervised learning (not with sparse matrices), you should scale your features. We'll be looking into it soon.

Parametric vs non parametric

- You might see a lot of definitions of these terms.
- A simple way to think about this is:
 - do you need to store at least $\mathcal{O}(n)$ worth of stuff to make predictions? If so, it's non-parametric.
- Non-parametric example: k -NN is a classic example of non-parametric models.
- Parametric example: decision stump
- If you want to know more about this, find some reading material [here](https://www.cs.ubc.ca/~schmidtm/Courses/340-F16/L6.pdf) (<https://www.cs.ubc.ca/~schmidtm/Courses/340-F16/L6.pdf>), [here](http://mlss.tuebingen.mpg.de/2015/slides/ghahramani/gp-neural-nets15.pdf) (<http://mlss.tuebingen.mpg.de/2015/slides/ghahramani/gp-neural-nets15.pdf>), and [here](https://machinelearningmastery.com/parametric-and-nonparametric-machine-learning-algorithms/) (<https://machinelearningmastery.com/parametric-and-nonparametric-machine-learning-algorithms/>).
- By the way, the terms "parametric" and "non-parametric" are often used differently by statisticians, see [here](https://help.xlstat.com/s/article/what-is-the-difference-between-a-parametric-and-a-nonparametric-test?language=en_US) (https://help.xlstat.com/s/article/what-is-the-difference-between-a-parametric-and-a-nonparametric-test?language=en_US) for more...

$\mathcal{O}(n)$ is referred to as big \mathcal{O} notation. It tells you how fast an algorithm is or how much storage space it requires. For example, in simple terms, if you have n examples and you need to store them all you can say that the algorithm requires $\mathcal{O}(n)$ worth of stuff.

Curse of dimensionality

- Affects all learners but especially bad for nearest-neighbour.
- k -NN usually works well when the number of dimensions d is small but things fall apart quickly as d goes up.
- If there are many irrelevant attributes, k -NN is hopelessly confused because all of them contribute to finding similarity between examples.
- With enough irrelevant attributes the accidental similarity swamps out meaningful similarity and k -NN is no better than random guessing.

In [42]:

```

1 from sklearn.datasets import make_classification
2
3 nfeats_accuracy = {"nfeats": [], "dummy_valid_accuracy": [], "KNN_valid_accuracy": []}
4 for n_feats in range(4, 2000, 100):
5     X, y = make_classification(n_samples=2000, n_features=n_feats, n_classes=2)
6     X_train, X_test, y_train, y_test = train_test_split(
7         X, y, test_size=0.2, random_state=123
8     )
9     dummy = DummyClassifier(strategy="most_frequent")
10    dummy_scores = cross_validate(dummy, X_train, y_train, return_train_score=True)
11
12    knn = KNeighborsClassifier()
13    scores = cross_validate(knn, X_train, y_train, return_train_score=True)
14    nfeats_accuracy["nfeats"].append(n_feats)
15    nfeats_accuracy["KNN_valid_accuracy"].append(np.mean(scores["test_score"]))
16    nfeats_accuracy["dummy_valid_accuracy"].append(np.mean(dummy_scores["test_score"]))

```

In [43]:

```
1 pd.DataFrame(nfeats_accuracy)
```

Out[43]:

	nfeats	dummy_valid_accuracy	KNN_valid_accuracy
0	4	0.510000	0.980000
1	104	0.502500	0.716250
2	204	0.501875	0.761250
3	304	0.503125	0.713750
4	404	0.508750	0.669375
5	504	0.500625	0.651250
6	604	0.505000	0.655000
7	704	0.505625	0.641250
8	804	0.506250	0.636875
9	904	0.500000	0.620000
10	1004	0.508750	0.610000
11	1104	0.503750	0.631875
12	1204	0.502500	0.600000
13	1304	0.504375	0.557500
14	1404	0.510625	0.568125
15	1504	0.503750	0.572500
16	1604	0.499375	0.576250
17	1704	0.503125	0.599375
18	1804	0.508125	0.551875
19	1904	0.506875	0.590625

Break (5 min)



Support Vector Machines (SVMs) with RBF kernel [[video](https://youtu.be/ic_zqOhi020) (https://youtu.be/ic_zqOhi020)]

- Very high-level overview
- Our goals here are
 - Use scikit-learn's SVM model.
 - Broadly explain the notion of support vectors.
 - Broadly explain the similarities and differences between \$k\$-NNs and SVM RBFs.
 - Explain how `C` and `gamma` hyperparameters control the fundamental tradeoff.

(Optional) RBF stands for radial basis functions. We won't go into what it means in this class. Refer to [this video](https://www.youtube.com/watch?v=Qc5IyLW_hns) (https://www.youtube.com/watch?v=Qc5IyLW_hns) if you want to know more.

Overview

- Another popular similarity-based algorithm is Support Vector Machines with RBF Kernel (SVM RBFs)

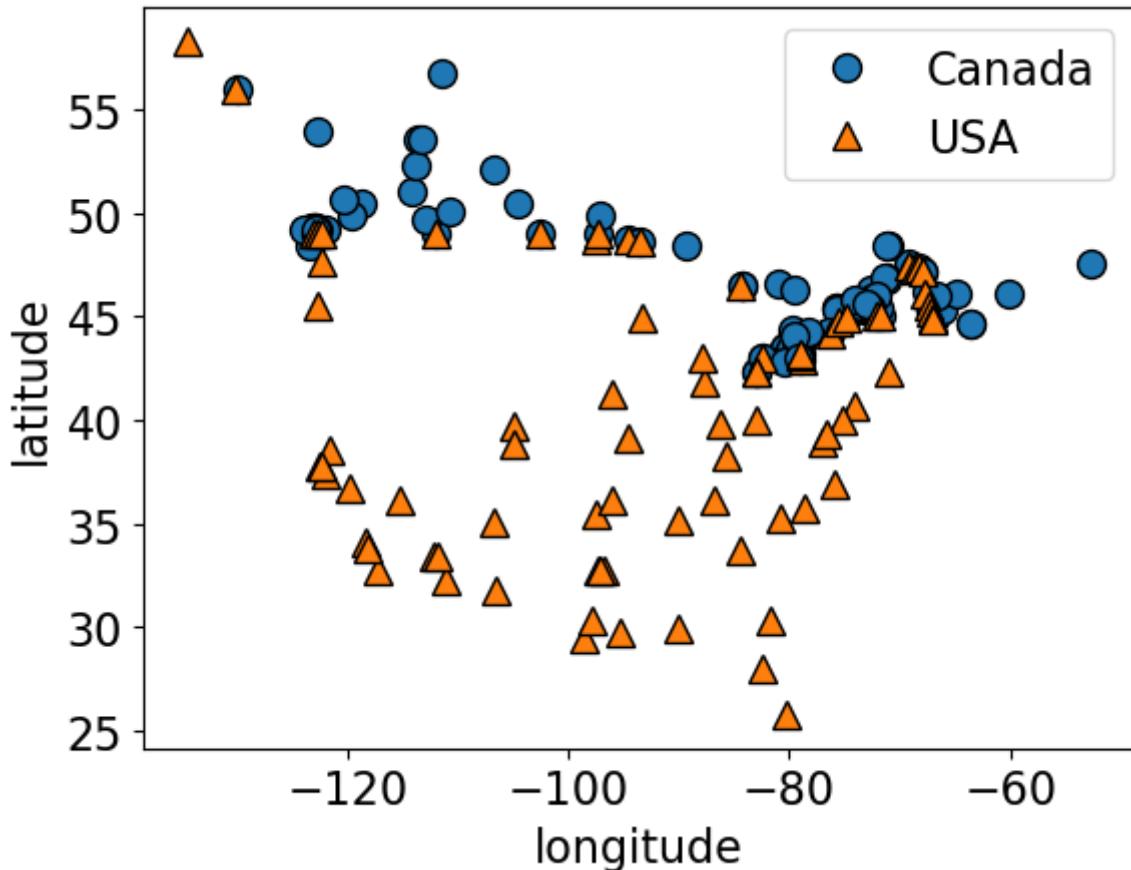
- Superficially, SVM RBFs are more like weighted k -NNs.
 - The decision boundary is defined by **a set of positive and negative examples and their weights** together with **their similarity measure**.
 - A test example is labeled positive if on average it looks more like positive examples than the negative examples.

- The primary difference between k -NNs and SVM RBFs is that
 - Unlike k -NNs, SVM RBFs only remember the key examples (support vectors). So it's more efficient than k -NN.
 - SVMs use a different similarity metric which is called a "kernel" in SVM land. A popular kernel is Radial Basis Functions (RBFs)
 - They usually perform better than k -NNs!

Let's explore SVM RBFs

Let's try SVMs on the cities dataset.

```
In [44]: 1 mglearn.discrete_scatter(X_cities.iloc[:, 0], X_cities.iloc[:, 1], y_ci
2 plt.xlabel("longitude")
3 plt.ylabel("latitude")
4 plt.legend(loc=1);
```



```
In [45]: 1 X_train, X_test, y_train, y_test = train_test_split(  
2         X_cities, y_cities, test_size=0.2, random_state=123  
3     )
```

```
In [46]: 1 knn = KNeighborsClassifier(n_neighbors=best_n_neighbours)  
2 scores = cross_validate(knn, X_train, y_train, return_train_score=True)  
3 print("Mean validation score %0.3f" % (np.mean(scores[ "test_score" ])))  
4 pd.DataFrame(scores)
```

Mean validation score 0.803

Out[46]:

	fit_time	score_time	test_score	train_score
0	0.005274	0.006239	0.794118	0.819549
1	0.002166	0.003047	0.764706	0.819549
2	0.001877	0.003529	0.727273	0.850746
3	0.001637	0.002546	0.787879	0.828358
4	0.001719	0.002608	0.939394	0.783582

```
In [47]: 1 from sklearn.svm import SVC  
2  
3 svm = SVC(gamma=0.01) # Ignore gamma for now  
4 scores = cross_validate(svm, X_train, y_train, return_train_score=True)  
5 print("Mean validation score %0.3f" % (np.mean(scores[ "test_score" ])))  
6 pd.DataFrame(scores)
```

Mean validation score 0.820

Out[47]:

	fit_time	score_time	test_score	train_score
0	0.006933	0.002283	0.823529	0.842105
1	0.002420	0.001686	0.823529	0.842105
2	0.002157	0.001490	0.727273	0.858209
3	0.002399	0.001730	0.787879	0.843284
4	0.002109	0.001452	0.939394	0.805970

Decision boundary of SVMs

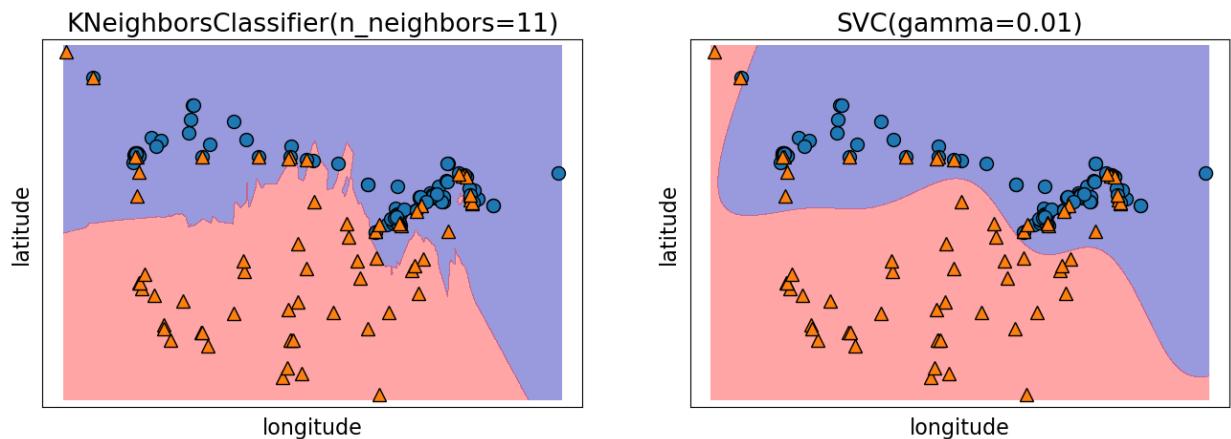
- We can think of SVM with RBF kernel as "smooth KNN".

In [48]:

```

1 fig, axes = plt.subplots(1, 2, figsize=(16, 5))
2
3 for clf, ax in zip([knn, svm], axes):
4     clf.fit(X_train, y_train)
5     mglearn.plots.plot_2d_separator(
6         clf, X_train.to_numpy(), fill=True, eps=0.5, ax=ax, alpha=0.4
7     )
8     mglearn.discrete_scatter(X_train.iloc[:, 0], X_train.iloc[:, 1], y_
9     ax.set_title(clf)
10    ax.set_xlabel("longitude")
11    ax.set_ylabel("latitude")

```



Support vectors

- Each training example either is or isn't a "support vector".
 - This gets decided during `fit`.
- **Main insight: the decision boundary only depends on the support vectors.**
- Let's look at the support vectors.

In [49]:

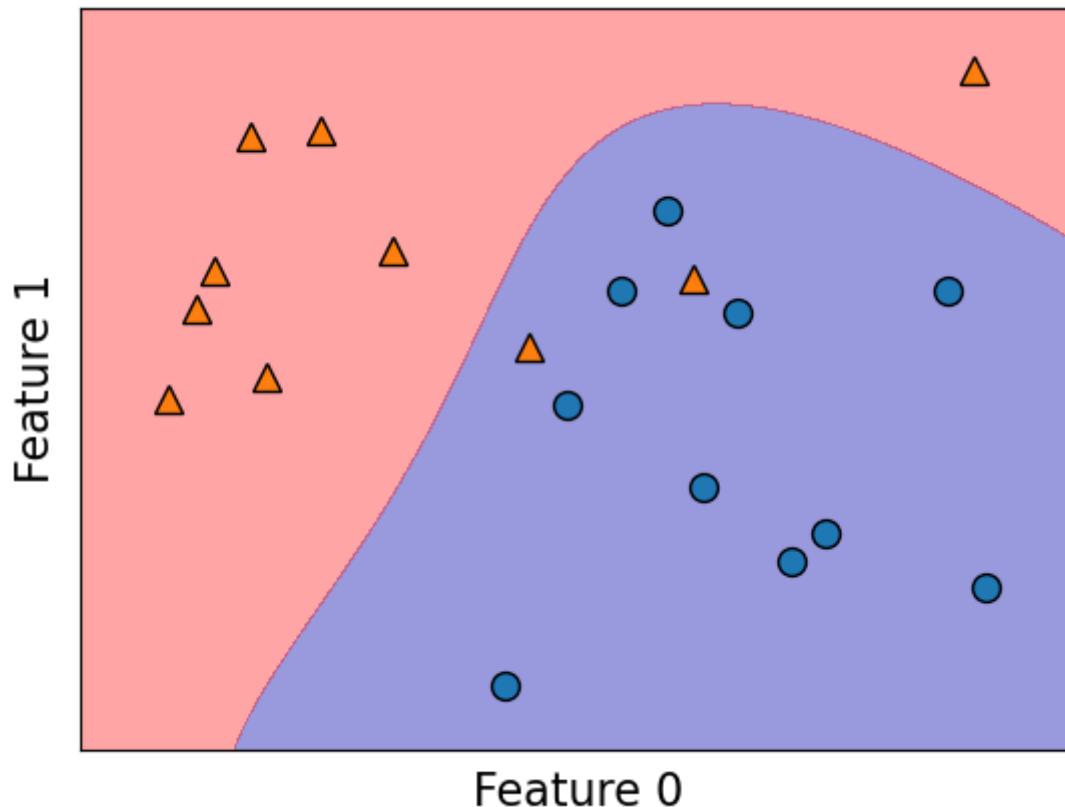
```

1 from sklearn.datasets import make_blobs
2
3 n = 20
4 n_classes = 2
5 X_toy, y_toy = make_blobs(
6     n_samples=n, centers=n_classes, random_state=300
7 ) # Let's generate some fake data

```

In [50]:

```
1 mglearn.discrete_scatter(X_toy[:, 0], X_toy[:, 1], y_toy)
2 plt.xlabel("Feature 0")
3 plt.ylabel("Feature 1")
4 svm = SVC(kernel="rbf", C=10, gamma=0.1).fit(X_toy, y_toy)
5 mglearn.plots.plot_2d_separator(svm, X_toy, fill=True, eps=0.5, alpha=0
```

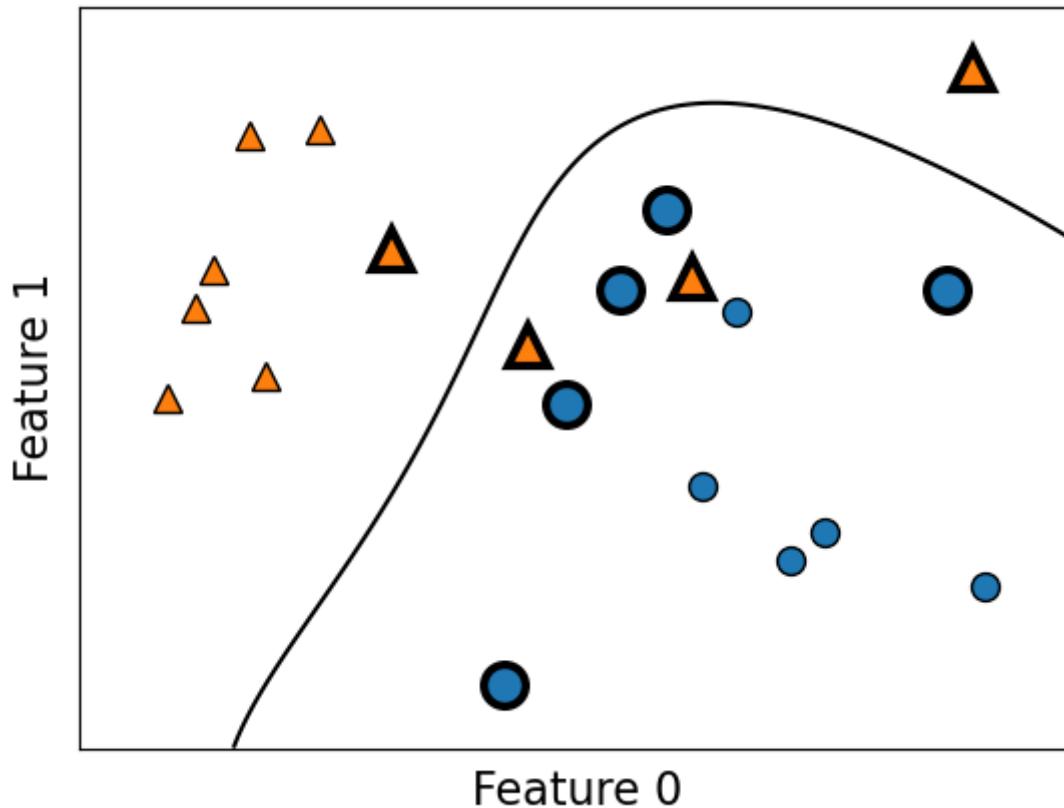


In [51]:

```
1 svm.support_
```

Out[51]: array([3, 8, 9, 14, 19, 1, 4, 6, 17], dtype=int32)

```
In [52]: 1 plot_support_vectors(svm, x_toy, y_toy)
```



The support vectors are the bigger points in the plot above.

Hyperparameters of SVM

- Key hyperparameters of `rbf` SVM are
 - `gamma`
 - `C`
- We are not equipped to understand the meaning of these parameters at this point but you are expected to describe their relation to the fundamental tradeoff.

See [scikit-learn's explanation of RBF SVM parameters \(\[https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html\]\(https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html\)\).](https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html)

Relation of `gamma` and the fundamental trade-off

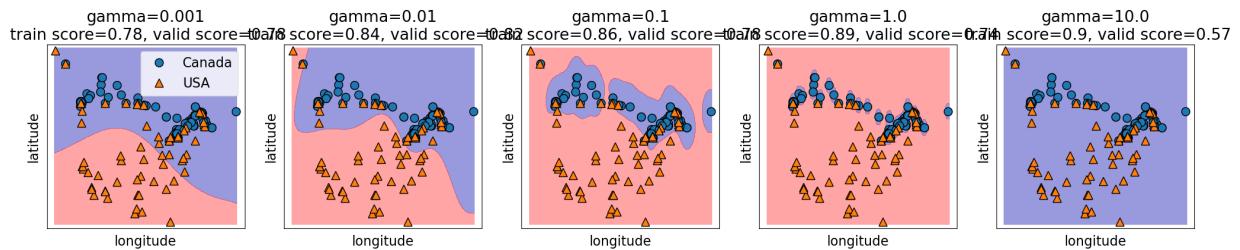
- `gamma` controls the complexity (fundamental trade-off), just like other hyperparameters we've seen.
 - larger `gamma` \rightarrow more complex
 - smaller `gamma` \rightarrow less complex

In [53]:

```

1 gamma = [0.001, 0.01, 0.1, 1.0, 10.0]
2 plot_svc_gamma(
3     gamma,
4     X_train.to_numpy(),
5     y_train.to_numpy(),
6     x_label="longitude",
7     y_label="latitude",
8 )

```



Relation of C and the fundamental trade-off

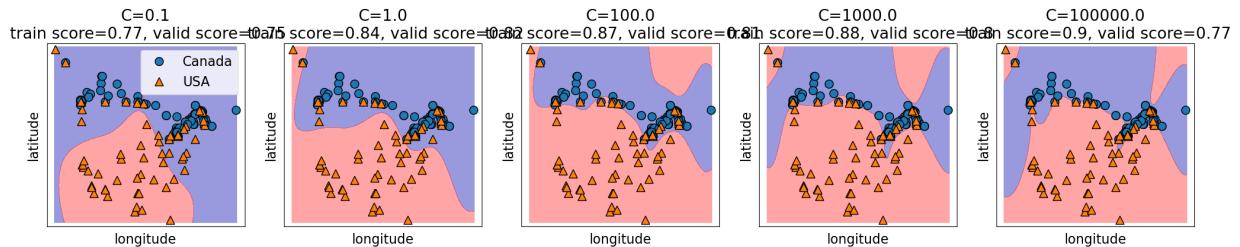
- C also affects the fundamental tradeoff
 - larger C \rightarrow more complex
 - smaller C \rightarrow less complex

In [54]:

```

1 C = [0.1, 1.0, 100.0, 1000.0, 100000.0]
2 plot_svc_C(
3     C, X_train.to_numpy(), y_train.to_numpy(), x_label="longitude", y_l
4 )

```



Search over multiple hyperparameters

- So far you have seen how to carry out search over a hyperparameter
- In the above case the best training error is achieved by the most complex model (large gamma , large C).
- Best validation error requires a hyperparameter search to balance the fundamental tradeoff.
 - In general we can't search them one at a time.
 - More on this in the Hyperparameter optimization lecture. But if you cannot wait till then, you may look up the following:
 - [sklearn.model_selection.GridSearchCV \(\[https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html\]\(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html\)\)](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)
 - [sklearn.model_selection.RandomizedSearchCV \(\[https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html\]\(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html\)\)](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html)

SVM Regressor

- Similar to KNNs, you can use SVMs for regression problems as well.
- See [sklearn.svm.SVR](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html) (<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html>) for more details.

?? Questions on SVM RBFs

Case study

A friend of yours shows you a SVM model they are working on, with very high accuracy on the training set (close to 100%).

Upon closer inspection, you find out that almost all the original samples have been selected as support vectors.

What can you guess about the values of `c` and `gamma` used to train this model?

Do you think this model will generalize well?

More practice questions

- Check out some more practice questions [here](https://ml-learn.mds.ubc.ca/en/module4) (<https://ml-learn.mds.ubc.ca/en/module4>).

Summary

- We have KNNs and SVMs as new supervised learning techniques in our toolbox.
- These are analogy-based learners and the idea is to assign nearby points the same label.
- Unlike decision trees, all features are equally important.
- Both can be used for classification or regression (much like the other methods we've seen).

Coming up:

Lingering questions:

- Are we ready to do machine learning on real-world datasets?
- What would happen if we use \$k\$-NNs or SVM RBFs on the spotify dataset from hw2?
- What happens if we have missing values in our data?
- What do we do if we have features with categories or string values?



In []:

1

CPSC 330

Applied Machine Learning

Lecture 5: Preprocessing and `sklearn` pipelines

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

In [1]:

```
1 import sys
2 import time
3
4 import matplotlib.pyplot as plt
5
6 %matplotlib inline
7 import numpy as np
8 import pandas as pd
9 from IPython.display import HTML
10
11 sys.path.append("../code/.")
12
13 import mglearn
14 from IPython.display import display
15 from plotting_functions import *
16
17 # Classifiers and regressors
18 from sklearn.dummy import DummyClassifier, DummyRegressor
19
20 # Preprocessing and pipeline
21 from sklearn.impute import SimpleImputer
22
23 # train test split and cross validation
24 from sklearn.model_selection import cross_val_score, cross_validate, train_test_split
25 from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
26 from sklearn.pipeline import Pipeline
27 from sklearn.preprocessing import (
28     MinMaxScaler,
29     OneHotEncoder,
30     OrdinalEncoder,
31     StandardScaler,
32 )
33 from sklearn.svm import SVC
34 from sklearn.tree import DecisionTreeClassifier
35 from utils import *
36
37 pd.set_option("display.max_colwidth", 200)
```

Announcements

- Homework 3 is out (Due date: Feb. 1st). Please start early!
- Homework 1 grades and solutions are out. Please do not share them with anyone or do not post them anywhere.

Learning outcomes

From this lecture, you will be able to

- explain the motivation for preprocessing in supervised machine learning;
- identify when to implement feature transformations such as imputation, scaling, and one-hot encoding in a machine learning model development pipeline;
- use `sklearn` transformers for applying feature transformations on your dataset;
- discuss the golden rule (a.k.a. don't look at the test set until the end) in the context of feature transformations;
- use `sklearn.pipeline.Pipeline` and `sklearn.pipeline.make_pipeline` to build a preliminary machine learning pipeline.

Motivation and big picture [[video](https://youtu.be/xx9HlmzORRk) (<https://youtu.be/xx9HlmzORRk>)]

- So far we have seen
 - Three ML models (decision trees, k -NNs, SVMs with RBF kernel)
 - ML fundamentals (train-validation-test split, cross-validation, the fundamental tradeoff, the golden rule)
- Are we ready to do machine learning on real-world datasets?
 - Very often real-world datasets need preprocessing before we use them to build ML models.

Example: k -nearest neighbours on the Spotify dataset

- In lab1 you used `DecisionTreeClassifier` to predict whether the user would like a particular song or not.
- Can we use a k -NN classifier for this task?
- Intuition: To predict whether the user likes a particular song or not (query point)
 - find the songs that are closest to the query point
 - let them vote on the target
 - take the majority vote as the target for the query point

In [2]:

```

1 spotify_df = pd.read_csv("../data/spotify.csv", index_col=0)
2 train_df, test_df = train_test_split(spotify_df, test_size=0.20, random_state=42)
3 X_train, y_train = (
4     train_df.drop(columns=["song_title", "artist", "target"]),
5     train_df["target"],
6 )
7 X_test, y_test = (
8     test_df.drop(columns=["song_title", "artist", "target"]),
9     test_df["target"],
10 )

```

```
In [3]: 1 dummy = DummyClassifier(strategy="most_frequent")
2 scores = cross_validate(dummy, X_train, y_train, return_train_score=True)
3 print("Mean validation score %0.3f" % (np.mean(scores["test_score"])))
4 pd.DataFrame(scores)
```

Mean validation score 0.508

```
Out[3]:   fit_time  score_time  test_score  train_score
0    0.004357    0.001195    0.507740    0.507752
1    0.000990    0.000347    0.507740    0.507752
2    0.000742    0.000344    0.507740    0.507752
3    0.000674    0.000306    0.506211    0.508133
4    0.000642    0.000300    0.509317    0.507359
```

```
In [4]: 1 knn = KNeighborsClassifier()
2 scores = cross_validate(knn, X_train, y_train, return_train_score=True)
3 print("Mean validation score %0.3f" % (np.mean(scores["test_score"])))
4 pd.DataFrame(scores)
```

Mean validation score 0.546

```
Out[4]:   fit_time  score_time  test_score  train_score
0    0.013952    0.018358    0.563467    0.717829
1    0.002404    0.009834    0.535604    0.721705
2    0.002058    0.008475    0.529412    0.708527
3    0.002010    0.008687    0.537267    0.721921
4    0.001937    0.008455    0.562112    0.711077
```

```
In [5]: 1 two_songs = X_train.sample(2, random_state=42)
2 two_songs
```

```
Out[5]:   acousticness  danceability  duration_ms  energy  instrumentalness  key  liveness  loudness  mode
842      0.229000        0.494     147893  0.666       0.000057    9  0.0469   -9.743
654      0.000289        0.771     227143  0.949       0.602000    8  0.5950   -4.712
```

```
In [6]: 1 euclidean_distances(two_songs)
```

```
Out[6]: array([[ 0.          , 79250.00543825],
   [79250.00543825,  0.          ]])
```

Let's consider only two features: duration_ms and tempo .

```
In [7]: 1 two_songs_subset = two_songs[["duration_ms", "tempo"]]
2 two_songs_subset
```

Out[7]:

	duration_ms	tempo
842	147893	140.832
654	227143	111.959

```
In [8]: 1 euclidean_distances(two_songs_subset)
```

Out[8]: array([[0. , 79250.00525962],
 [79250.00525962, 0.]])

Do you see any problem?

- The distance is completely dominated by the features with larger values
- The features with smaller values are being ignored.
- Does it matter?
 - Yes! Scale is based on how data was collected.
 - Features on a smaller scale can be highly informative and there is no good reason to ignore them.
 - We want our model to be robust and not sensitive to the scale.
- Was this a problem for decision trees?

Scaling using scikit-learn's [StandardScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html) (<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>)

- We'll use scikit-learn's [StandardScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html) (<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>), which is a transformer.
- Only focus on the syntax for now. We'll talk about scaling in a bit.

In [9]:

```

1 from sklearn.preprocessing import StandardScaler
2
3 scaler = StandardScaler() # create feature transformer object
4 scaler.fit(X_train) # fitting the transformer on the train split
5 X_train_scaled = scaler.transform(X_train) # transforming the train split
6 X_test_scaled = scaler.transform(X_test) # transforming the test split
7 pd.DataFrame(X_train_scaled, columns=X_train.columns).head()

```

Out[9]:

	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudn
0	-0.697633	-0.194548	-0.398940	-0.318116	-0.492359	1.275623	-0.737898	0.395
1	-0.276291	0.295726	-0.374443	-0.795552	0.598355	-1.487342	-0.438792	-0.052
2	-0.599540	1.110806	-0.376205	-0.946819	-0.492917	0.446734	-0.399607	-0.879
3	-0.307150	1.809445	-0.654016	-1.722063	-0.492168	0.170437	-0.763368	-1.460
4	-0.634642	0.491835	-0.131344	1.057468	2.723273	0.170437	-0.458384	-0.175

fit and transform paradigm for transformers

- sklearn uses fit and transform paradigms for feature transformations.
- We fit the transformer **on the train split** and then transform the train split as well as the test split.
- We apply the same transformations on the test split.

sklearn API summary: estimators

Suppose `model` is a classification or regression model.

```

model.fit(X_train, y_train)
X_train_predictions = model.predict(X_train)
X_test_predictions = model.predict(X_test)

```

sklearn API summary: transformers

Suppose `transformer` is a transformer used to change the input representation, for example, to tackle missing values or to scale numeric features.

```

transformer.fit(X_train, [y_train])
X_train_transformed = transformer.transform(X_train)
X_test_transformed = transformer.transform(X_test)

```

- You can pass `y_train` in `fit` but it's usually ignored. It allows you to pass it just to be consistent with usual usage of `sklearn`'s `fit` method.
- You can also carry out fitting and transforming in one call using `fit_transform`. But be mindful to use it only on the train split and **not** on the test split.

- Do you expect `DummyClassifier` results to change after scaling the data?
- Let's check whether scaling makes any difference for k -NNs.

In [10]:

```

1 knn_unscaled = KNeighborsClassifier()
2 knn_unscaled.fit(X_train, y_train)
3 print("Train score: %0.3f" % (knn_unscaled.score(X_train, y_train)))
4 print("Test score: %0.3f" % (knn_unscaled.score(X_test, y_test)))

```

Train score: 0.726
Test score: 0.552

In [11]:

```

1 knn_scaled = KNeighborsClassifier()
2 knn_scaled.fit(X_train_scaled, y_train)
3 print("Train score: %0.3f" % (knn_scaled.score(X_train_scaled, y_train)))
4 print("Test score: %0.3f" % (knn_scaled.score(X_test_scaled, y_test)))

```

Train score: 0.798
Test score: 0.686

- The scores with scaled data are better compared to the unscaled data in case of k -NNs.
- I am not carrying out cross-validation here for a reason that we'll look into soon.
- Note that I am a bit sloppy here and using the test set several times for teaching purposes. But when you build an ML pipeline, please do assessment on the test set only once.

Common preprocessing techniques

Some commonly performed feature transformation include:

- Imputation: Tackling missing values
- Scaling: Scaling of numeric features
- One-hot encoding: Tackling categorical variables

We could have one lecture on each of them! In this lesson our goal is to get familiar with them so that we can use them to build ML pipelines.

In the next part of this lecture, we'll build an ML pipeline using [California housing prices regression dataset \(<https://www.kaggle.com/harrywang/housing>\)](https://www.kaggle.com/harrywang/housing). In the process, we will talk about different feature transformations and how can we apply them so that we do not violate the golden rule.

Imputation and scaling [[video](#) (<https://youtu.be/G2IXbVzKlt8>)]

Dataset, splitting, and baseline

We'll be working on [California housing prices regression dataset](#) (<https://www.kaggle.com/harrywang/housing>) to demonstrate these feature transformation techniques. The task is to predict median house values in Californian districts, given a number of features from these districts. If you are running the notebook on your own, you'll have to download the data and put it in the data directory.

In [12]:

```
1 housing_df = pd.read_csv("../data/housing.csv")
2 train_df, test_df = train_test_split(housing_df, test_size=0.1, random_
3
4 train_df.head()
```

Out[12]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
6051	-117.75	34.04	22.0	2948.0	636.0	2600.0	602.0
20113	-119.57	37.94	17.0	346.0	130.0	51.0	20.0
14289	-117.13	32.74	46.0	3355.0	768.0	1457.0	708.0
13665	-117.31	34.02	18.0	1634.0	274.0	899.0	285.0
14471	-117.23	32.88	18.0	5566.0	1465.0	6303.0	1458.0

Some column values are mean/median but some are not.

Let's add some new features to the dataset which could help predicting the target:
`median_house_value`.

In [13]:

```

1 train_df = train_df.assign(
2     rooms_per_household=train_df["total_rooms"] / train_df["households"]
3 )
4 test_df = test_df.assign(
5     rooms_per_household=test_df["total_rooms"] / test_df["households"]
6 )
7
8 train_df = train_df.assign(
9     bedrooms_per_household=train_df["total_bedrooms"] / train_df["households"]
10)
11 test_df = test_df.assign(
12     bedrooms_per_household=test_df["total_bedrooms"] / test_df["households"]
13)
14
15 train_df = train_df.assign(
16     population_per_household=train_df["population"] / train_df["households"]
17)
18 test_df = test_df.assign(
19     population_per_household=test_df["population"] / test_df["households"]
20)

```

In [14]:

```
1 train_df.head()
```

Out[14]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
6051	-117.75	34.04	22.0	2948.0	636.0	2600.0	602.0
20113	-119.57	37.94	17.0	346.0	130.0	51.0	20.0
14289	-117.13	32.74	46.0	3355.0	768.0	1457.0	708.0
13665	-117.31	34.02	18.0	1634.0	274.0	899.0	285.0
14471	-117.23	32.88	18.0	5566.0	1465.0	6303.0	1458.0

When is it OK to do things before splitting?

- Here it would have been OK to add new features before splitting because we are not using any global information about the feature values (e.g. the mean of a feature across rows), but only looking at one row at a time.
- But just to be safe and to avoid accidentally breaking the golden rule, it's better to do it after splitting.
- Question: Should we remove `total_rooms`, `total_bedrooms`, and `population` columns?
 - Probably. But I am keeping them in this lecture. You could experiment with removing them and examine whether results change.

Exploratory Data Analysis (EDA)

In [15]: 1 train_df.head()

Out[15]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
6051	-117.75	34.04	22.0	2948.0	636.0	2600.0	602.0
20113	-119.57	37.94	17.0	346.0	130.0	51.0	20.0
14289	-117.13	32.74	46.0	3355.0	768.0	1457.0	708.0
13665	-117.31	34.02	18.0	1634.0	274.0	899.0	285.0
14471	-117.23	32.88	18.0	5566.0	1465.0	6303.0	1458.0

The feature scales are quite different.

In [16]: 1 train_df.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18576 entries, 6051 to 19966
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   longitude        18576 non-null   float64
 1   latitude         18576 non-null   float64
 2   housing_median_age 18576 non-null   float64
 3   total_rooms      18576 non-null   float64
 4   total_bedrooms   18391 non-null   float64
 5   population       18576 non-null   float64
 6   households       18576 non-null   float64
 7   median_income    18576 non-null   float64
 8   median_house_value 18576 non-null   float64
 9   ocean_proximity  18576 non-null   object  
 10  rooms_per_household 18576 non-null   float64
 11  bedrooms_per_household 18391 non-null   float64
 12  population_per_household 18576 non-null   float64
dtypes: float64(12), object(1)
memory usage: 2.0+ MB
```

We have one categorical feature and all other features are numeric features.

In [17]: 1 train_df.describe()

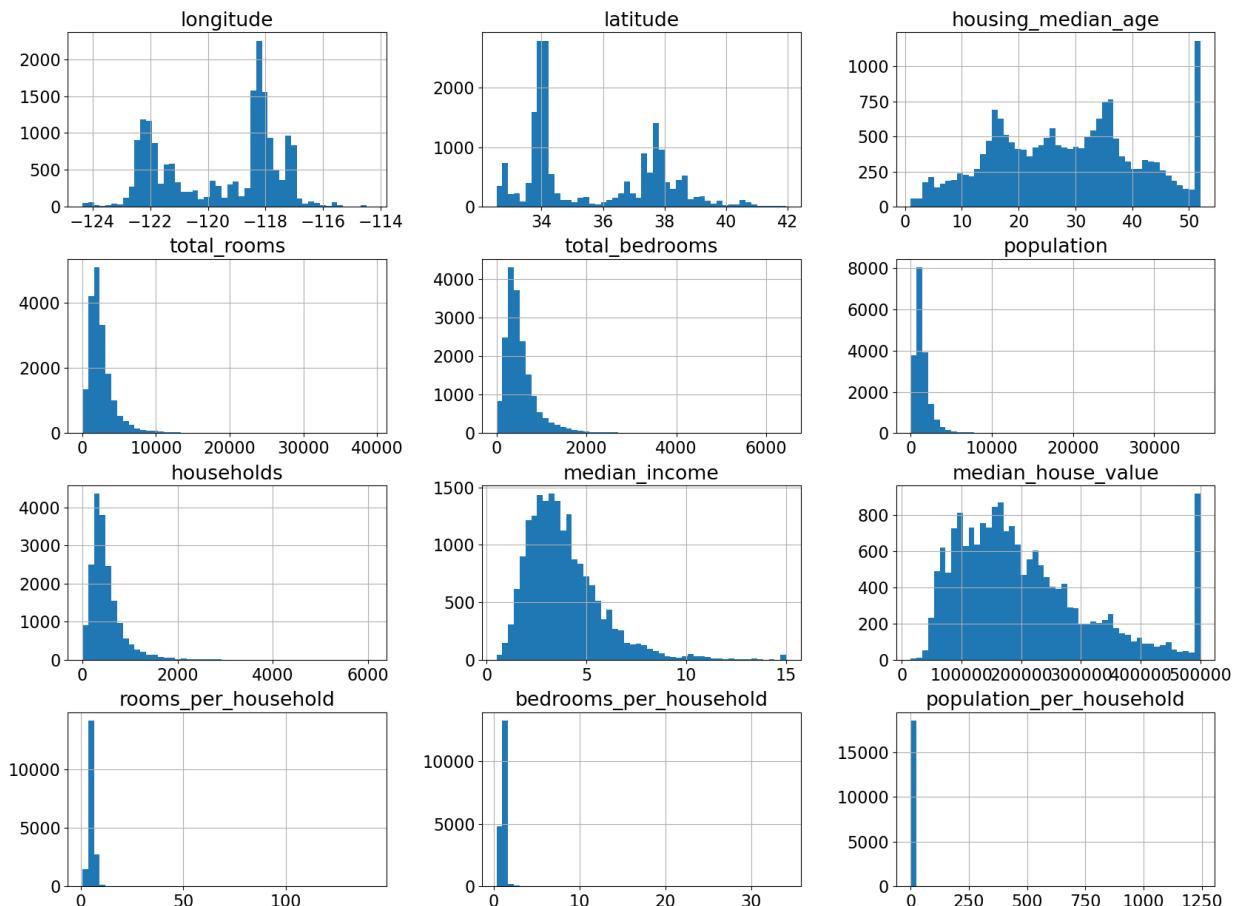
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
count	18576.000000	18576.000000	18576.000000	18576.000000	18391.000000	18576.000000
mean	-119.565888	35.627966	28.622255	2635.749677	538.229786	1428.578161
std	1.999622	2.134658	12.588307	2181.789934	421.805266	1141.664801
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000
25%	-121.790000	33.930000	18.000000	1449.000000	296.000000	788.000000
50%	-118.490000	34.250000	29.000000	2127.000000	435.000000	1167.000000
75%	-118.010000	37.710000	37.000000	3145.000000	647.000000	1727.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000

- Seems like total_bedrooms column has some missing values.
- This must have affected our new feature bedrooms_per_household as well.

In [18]: 1 housing_df["total_bedrooms"].isnull().sum()

Out[18]: 207

In [19]: 1 ## (optional)
2 train_df.hist(bins=50, figsize=(20, 15));

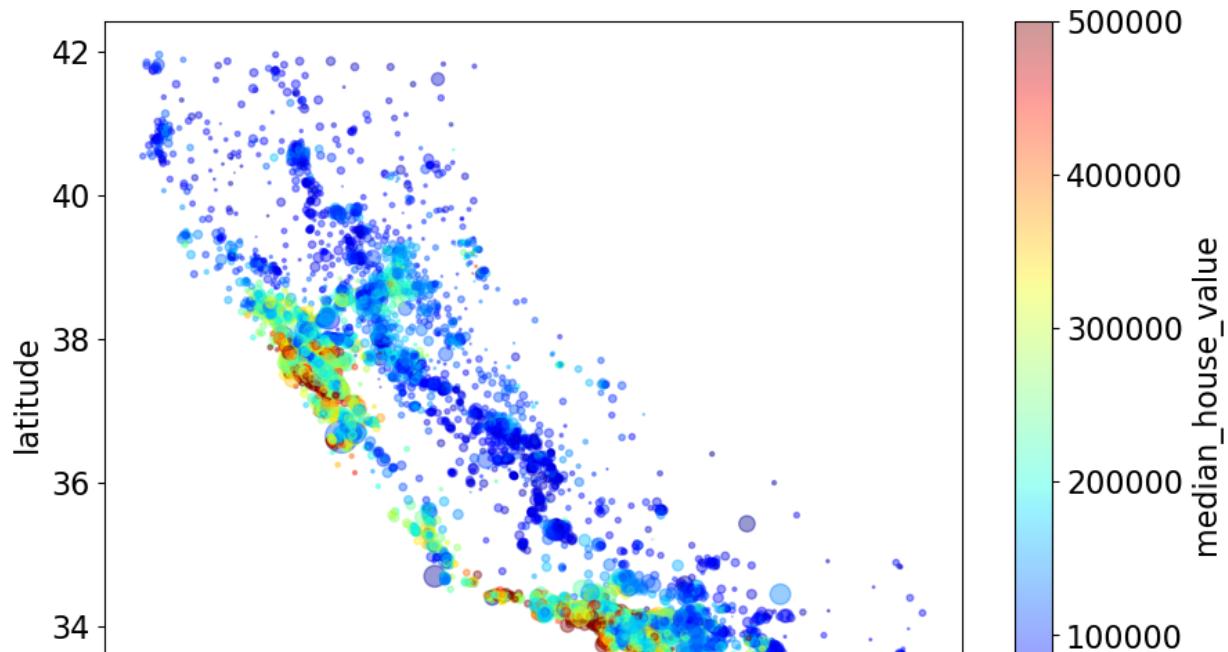


In [20]:

```

1 ## (optional)
2 train_df.plot(
3     kind="scatter",
4     x="longitude",
5     y="latitude",
6     alpha=0.4,
7     s=train_df[ "population" ] / 100,
8     figsize=(10, 7),
9     c="median_house_value",
10    cmap=plt.get_cmap("jet"),
11    colorbar=True,
12    sharex=False,
13 );

```



Was that all the transformations we need to apply to the dataset?

Here is what we see from the exploratory data analysis (EDA).

- Some missing values in `total_bedrooms` column.
- Scales are quite different across columns.
- Categorical variable `ocean_proximity`.

Read about [preprocessing techniques implemented in scikit-learn](https://scikit-learn.org/stable/modules/preprocessing.html) (<https://scikit-learn.org/stable/modules/preprocessing.html>).

In [21]:

```

1 # We are droping the categorical variable ocean_proximity for now. We'll
2 X_train = train_df.drop(columns=[ "median_house_value", "ocean_proximity" ])
3 y_train = train_df[ "median_house_value" ]
4
5 X_test = test_df.drop(columns=[ "median_house_value", "ocean_proximity" ])
6 y_test = test_df[ "median_house_value" ]

```

Let's first run our baseline model DummyRegressor

```
In [22]: 1 results_dict = {} # dictionary to store our results for different mode
```

```
In [23]: 1 def mean_std_cross_val_scores(model, X_train, y_train, **kwargs):
2     """
3         Returns mean and std of cross validation
4
5         Parameters
6         -----
7         model :
8             scikit-learn model
9         X_train : numpy array or pandas DataFrame
10            X in the training data
11         y_train :
12            y in the training data
13
14         Returns
15         -----
16            pandas Series with mean scores from cross_validation
17        """
18
19         scores = cross_validate(model, X_train, y_train, **kwargs)
20
21         mean_scores = pd.DataFrame(scores).mean()
22         std_scores = pd.DataFrame(scores).std()
23         out_col = []
24
25         for i in range(len(mean_scores)):
26             out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
27
28     return pd.Series(data=out_col, index=mean_scores.index)
```

```
In [24]: 1 dummy = DummyRegressor(strategy="median")
2 results_dict["dummy"] = mean_std_cross_val_scores(
3     dummy, X_train, y_train, return_train_score=True
4 )
```

```
In [25]: 1 pd.DataFrame(results_dict)
```

Out[25]:

	dummy
fit_time	0.002 (+/- 0.001)
score_time	0.001 (+/- 0.000)
test_score	-0.055 (+/- 0.012)
train_score	-0.055 (+/- 0.001)

Imputation

```
In [27]: 1 knn = KNeighborsRegressor()  
2 knn.fit(X_train, y_train)
```

```

-----
-->      raise ValueError(
          ^
ValueError                                Traceback (most recent call last)
t)
Cell In[27], line 2
      1 knn = KNeighborsRegressor()
----> 2 knn.fit(X_train, y_train)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/neigh
bors/_regression.py:210, in KNeighborsRegressor.fit(self, X, y)
   191     """Fit the k-nearest neighbors regressor from the training data
   192     set.
   193     Parameters
   (...)

 206         The fitted k-nearest neighbors regressor.
 207     """
 208     self.weights = _check_weights(self.weights)
--> 210     return self._fit(X, y)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/neigh
bors/_base.py:407, in NeighborsBase._fit(self, X, y)
  405     if self._get_tags()["requires_y"]:
  406         if not isinstance(X, (KDTree, BallTree, NeighborsBase)):
--> 407             X, y = self._validate_data(
  408                 X, y, accept_sparse="csr", multi_output=True, order
="C"
  409             )
  410     if is_classifier(self):
  411         # Classification targets require a specific format
  412         if y.ndim == 1 or y.ndim == 2 and y.shape[1] == 1:

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/base.
py:596, in BaseEstimator._validate_data(self, X, y, reset, validate_separ
ately, **check_params)
  594         y = check_array(y, input_name="y", **check_y_params)
  595     else:
--> 596         X, y = check_X_y(X, y, **check_params)
  597     out = X, y
  599 if not no_val_X and check_params.get("ensure_2d", True):

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/util
s/validation.py:1074, in check_X_y(X, y, accept_sparse, accept_large_spar
se, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, multi_outp
ut, ensure_min_samples, ensure_min_features, y_numeric, estimator)
 1069         estimator_name = _check_estimator_name(estimator)
 1070     raise ValueError(
 1071         f"{estimator_name} requires y to be passed, but the targe
t y is None"
 1072     )
-> 1074 X = check_array(
 1075     X,
 1076     accept_sparse=accept_sparse,
 1077     accept_large_sparse=accept_large_sparse,
 1078     dtype=dtype,
 1079     order=order,
 1080     copy=copy,

```

```

1081     force_all_finite=force_all_finite,
1082     ensure_2d=ensure_2d,
1083     allow_nd=allow_nd,
1084     ensure_min_samples=ensure_min_samples,
1085     ensure_min_features=ensure_min_features,
1086     estimator=estimator,
1087     input_name="X",
1088 )
1090 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric, estimator=estimator)
1092 check_consistent_length(X, y)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/util
s/validation.py:899, in check_array(array, accept_sparse, accept_large_sp
arse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_m
in_samples, ensure_min_features, estimator, input_name)
893         raise ValueError(
894             "Found array with dim %d. %s expected <= 2."
895             % (array.ndim, estimator_name)
896         )
898     if force_all_finite:
--> 899         _assert_all_finite(
900             array,
901             input_name=input_name,
902             estimator_name=estimator_name,
903             allow_nan=force_all_finite == "allow-nan",
904         )
906 if ensure_min_samples > 0:
907     n_samples = _num_samples(array)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/util
s/validation.py:146, in _assert_all_finite(X, allow_nan, msg_dtype, estim
ator_name, input_name)
124         if (
125             not allow_nan
126             and estimator_name
(...):
130             # Improve the error message on how to handle missing
values in
131             # scikit-learn.
132             msg_err += (
133                 f"\n{estimator_name} does not accept missing val
ues"
134                 " encoded as NaN natively. For supervised learnin
g, you might want"
(...):
144                 "#estimators-that-handle-nan-values"
145             )
--> 146         raise ValueError(msg_err)
148 # for object dtype data, we only check for NaNs (GH-13254)
149 elif X.dtype == np.dtype("object") and not allow_nan:

ValueError: Input X contains NaN.
KNeighborsRegressor does not accept missing values encoded as NaN natively
y. For supervised learning, you might want to consider sklearn.ensemble.H
istGradientBoostingClassifier and Regressor which accept missing values e
ncoded as NaNs natively. Alternatively, it is possible to preprocess the

```

data, for instance by using an imputer transformer in a pipeline or drop samples with missing values. See <https://scikit-learn.org/stable/modules/impute.html> (<https://scikit-learn.org/stable/modules/impute.html>) You can find a list of all estimators that handle NaN values at the following page: <https://scikit-learn.org/stable/modules/impute.html#estimators-that-handle-nan-values> (<https://scikit-learn.org/stable/modules/impute.html#estimators-that-handle-nan-values>)

What's the problem?

`ValueError: Input contains NaN, infinity or a value too large for dtype('float64').`

- The classifier is not able to deal with missing values (NaNs).
- What are possible ways to deal with the problem?
 - Delete the rows?
 - Replace them with some reasonable values?

- `SimpleImputer` is a transformer in `sklearn` to deal with this problem. For example:
 - You can impute missing values in categorical columns with the most frequent value.
 - You can impute the missing values in numeric columns with the mean or median of the column.

In [28]: 1 `X_train.sort_values("bedrooms_per_household")`

Out[28]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
20248	-119.23	34.25		28.0	26.0	3.0	29.0
12649	-121.47	38.51		52.0	20.0	4.0	74.0
3125	-117.76	35.22		4.0	18.0	3.0	8.0
12138	-117.22	33.87		16.0	56.0	7.0	39.0
8219	-118.21	33.79		33.0	32.0	18.0	96.0
...
4591	-118.28	34.06		42.0	2472.0	NaN	3795.0
19485	-120.98	37.66		10.0	934.0	NaN	401.0
6962	-118.05	33.99		38.0	1619.0	NaN	886.0
14970	-117.01	32.74		31.0	3473.0	NaN	2098.0
7763	-118.10	33.91		36.0	726.0	NaN	490.0

18576 rows × 11 columns

```
In [29]: 1 X_train.shape
          2 X_test.shape
```

Out[29]: (2064, 11)

```
In [32]: 1 imputer = SimpleImputer(strategy="median")
          2 imputer.fit(X_train)
          3 X_train_imp = imputer.transform(X_train)
          4 X_test_imp = imputer.transform(X_test)
```

- Let's check whether the NaN values have been replaced or not
- Note that `imputer.transform` returns an numpy array and not a dataframe

```
In [37]: 1 df = pd.DataFrame(X_train_imp, columns=X_train.columns, index=X_train.i
          2 df.loc[7763]
          3 # df.loc[7763]
          4 df.isnull().sum()
```

Out[37]:

longitude	0
latitude	0
housing_median_age	0
total_rooms	0
total_bedrooms	0
population	0
households	0
median_income	0
rooms_per_household	0
bedrooms_per_household	0
population_per_household	0
<code>dtype: int64</code>	

- Now the k -NN runs!
- The training error is bad though...

```
In [38]: 1 knn = KNeighborsRegressor()
          2 knn.fit(X_train_imp, y_train)
          3 knn.score(X_train_imp, y_train)
```

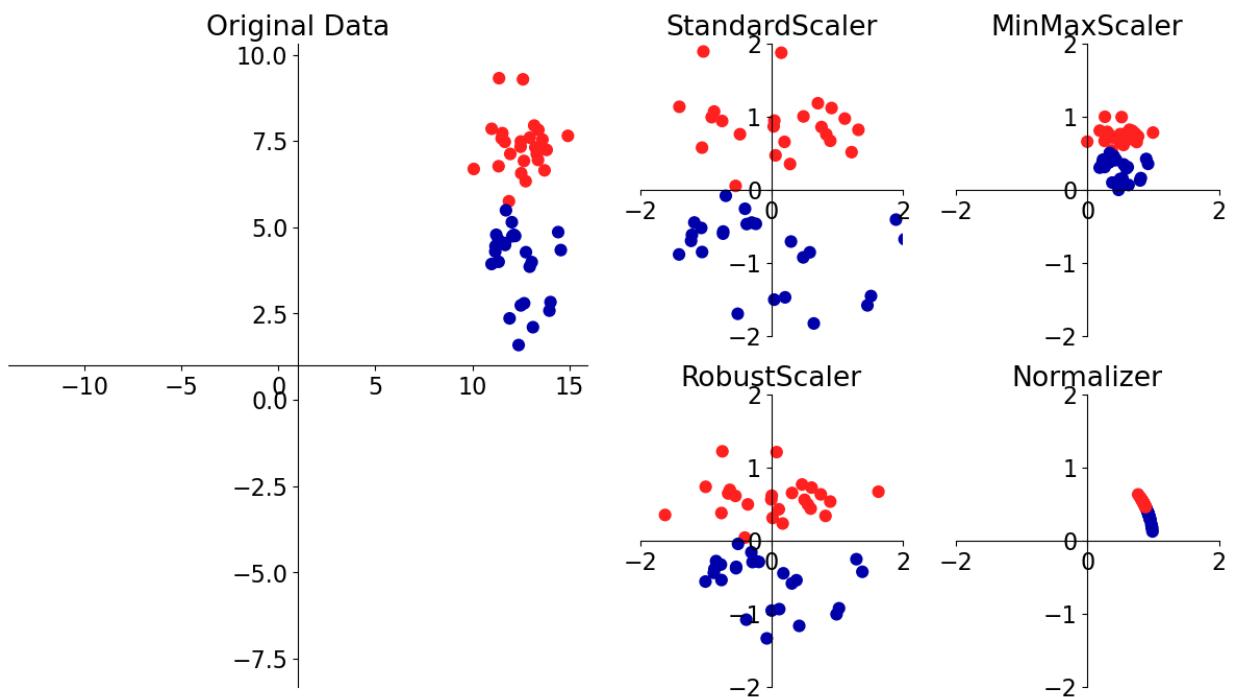
Out[38]: 0.5085407150708963

Scaling

- This problem affects a large number of ML methods.
- A number of approaches to this problem. We are going to look into the two most popular ones.

Approach	What it does	How to update X (but see below!)
normalization	sets range to [0, 1]	$X \leftarrow \frac{X - \min(X, axis=0)}{\max(X, axis=0)}$ <code>MinMaxScaler()</code> (https://scikit-learn.org/stable/modules/preprocessing.html#min-max-scaling)

```
In [39]: 1 # [source] (https://amueller.github.io/COMS4995-s19/slides/aml-05-preproc
          2 mglearn.plots.plot_scaling()
```



```
In [40]: 1 from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

In [41]:

```

1 scaler = StandardScaler()
2 X_train_scaled = scaler.fit_transform(X_train_imp) # fit once
3 X_test_scaled = scaler.transform(X_test_imp) # Transform test set
4 pd.DataFrame(X_train_scaled, columns=X_train.columns)

```

Out[41]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
0	0.908140	-0.743917	-0.526078	0.143120	0.235339	1.026092	0.2661
1	-0.002057	1.083123	-0.923283	-1.049510	-0.969959	-1.206672	-1.2533
2	1.218207	-1.352930	1.380504	0.329670	0.549764	0.024896	0.5428
3	1.128188	-0.753286	-0.843842	-0.459154	-0.626949	-0.463877	-0.5614
4	1.168196	-1.287344	-0.843842	1.343085	2.210026	4.269688	2.5009
...
18571	0.733102	-0.804818	0.586095	-0.875337	-0.243446	-0.822136	-0.9661
18572	1.163195	-1.057793	-1.161606	0.940194	0.609314	0.882438	0.7282
18573	-1.097293	0.797355	-1.876574	0.695434	0.433046	0.881563	0.5141
18574	-1.437367	1.008167	1.221622	-0.499947	-0.484029	-0.759944	-0.4544
18575	0.242996	0.272667	-0.684960	-0.332190	-0.353018	-0.164307	-0.3969

18576 rows × 11 columns

In [42]:

```

1 knn = KNeighborsRegressor()
2 knn.fit(X_train_scaled, y_train)
3 knn.score(X_train_scaled, y_train)

```

Out[42]: 0.8090877831586284

- Big difference in the KNN training performance after scaling the data.
- But we saw last week that training score doesn't tell us much. We should look at the cross-validation score.

?? Questions for class discussion

True/False on scaling and imputation

1. StandardScaler ensures a fixed range (i.e., minimum and maximum values) for the features.
2. StandardScaler calculates mean and standard deviation for each feature separately.
3. In general, it's a good idea to apply scaling on numeric features before training k -NN or SVM RBF models.
4. After normalization, all numerical features in training and test set will have values between 0 and 1.
5. It is OK to modify features before splitting into training and testing set if the modification is row-wise.

1. False
2. True
3. True
4. False (not in the test set)
5. True (but we don't recommend it)

More True/False on scaling and imputation

4. The transformers such as StandardScaler or SimpleImputer in scikit-learn return a dataframe with transformed features.
5. The transformed feature values might be hard to interpret for humans.
6. After applying SimpleImputer, the transformed data has a different shape than the original data.

Consider a toy data with the following two columns. If you apply StandardScaler on this data, both columns A and B will end up being identical. True or False?

```
In [ ]: 1 ex_test_cols = np.array([[10, -2], [20, -1], [30, 0], [40, 1], [50, 2]])
2 ex_test_df = pd.DataFrame(data=ex_test_cols, columns=["A", "B"])
3 ex_test_df
```

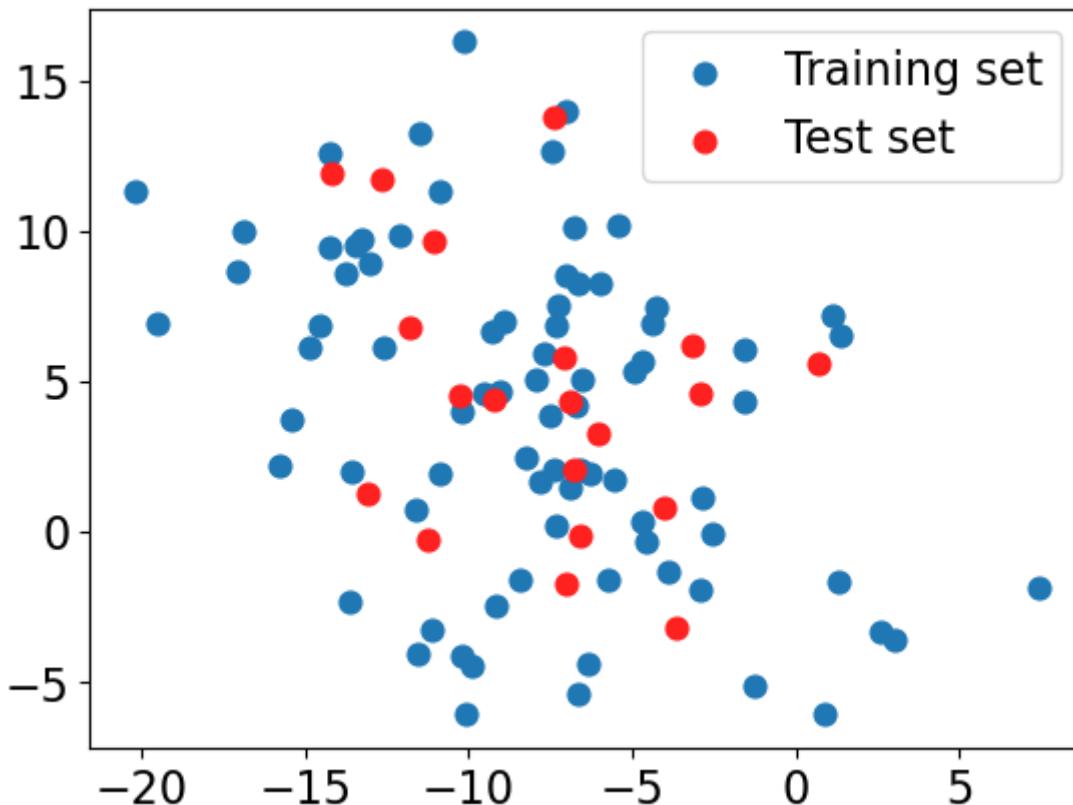
Let's create some synthetic data.

In [43]:

```

1 from sklearn.datasets import make_blobs, make_classification
2
3 # make synthetic data
4 X, y = make_blobs(n_samples=100, centers=3, random_state=12, cluster_std=1)
5 # split it into training and test sets
6 X_train_toy, X_test_toy, y_train_toy, y_test_toy = train_test_split(
7     X, y, random_state=5, test_size=0.2
8 )
9 plt.scatter(X_train_toy[:, 0], X_train_toy[:, 1], label="Training set",
10 plt.scatter(
11     X_test_toy[:, 0], X_test_toy[:, 1], color=mlearn.cm2(1), label="Te
12 )
13 plt.legend(loc="upper right");

```



Let's transform the data using `StandardScaler` and examine how the data looks like.

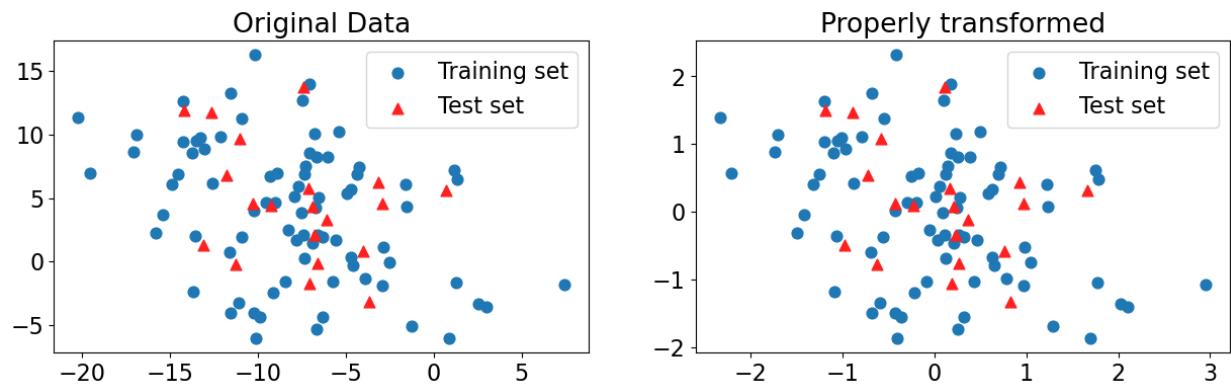
In [44]:

```

1 scaler = StandardScaler()
2 train_transformed = scaler.fit_transform(X_train_toy)
3 test_transformed = scaler.transform(X_test_toy)

```

```
In [45]: 1 plot_original_scaled(X_train_toy, X_test_toy, train_transformed, test_t
```



```
In [46]: 1 knn = KNeighborsClassifier()
2 knn.fit(train_transformed, y_train_toy)
3 print(f"Training score: {knn.score(train_transformed, y_train_toy):.2f}")
4 print(f"Test score: {knn.score(test_transformed, y_test_toy):.2f}")
```

Training score: 0.75

Test score: 0.55

Bad methodology 1: Scaling the data separately (for class discussion)

```
In [48]: 1 # DO NOT DO THIS! For illustration purposes only.
2 scaler = StandardScaler()
3 scaler.fit(X_train_toy)
4 train_scaled = scaler.transform(X_train_toy)
5
6 scaler = StandardScaler() # Creating a separate object for scaling tes
7 scaler.fit(X_test_toy) # Calling fit on the test data
8 test_scaled = scaler.transform(
9     X_test_toy
10 ) # Transforming the test data using the scaler fit on test data
11
12 knn = KNeighborsClassifier()
13 knn.fit(train_scaled, y_train_toy)
14 print(f"Training score: {knn.score(train_scaled, y_train_toy):.2f}")
15 print(f"Test score: {knn.score(test_scaled, y_test_toy):.2f}")
```

Training score: 0.75

Test score: 0.50

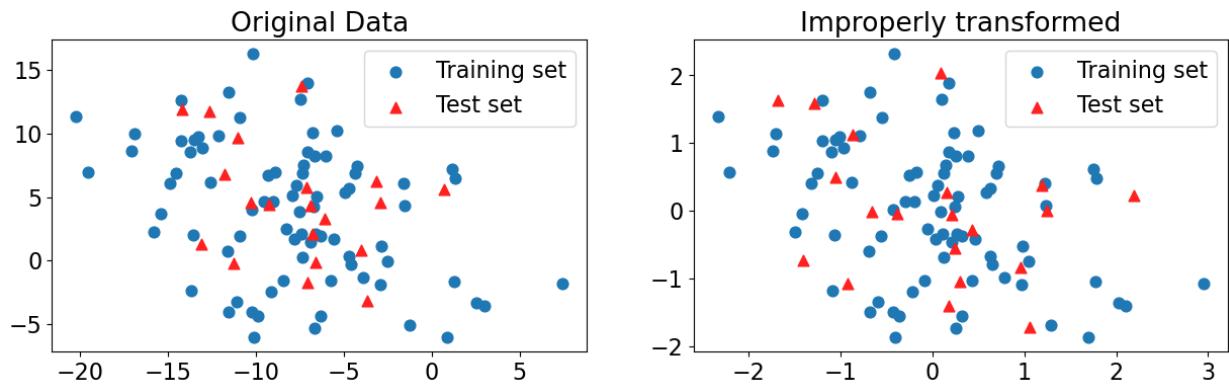
- Is anything wrong in methodology 1? If yes, what is it?

In [49]:

```

1 plot_original_scaled(
2     X_train_toy,
3     X_test_toy,
4     train_scaled,
5     test_scaled,
6     title_transformed="Improperly transformed",
7 )

```



Bad methodology 2: Scaling the data together (for class discussion)

In [51]:

```
1 X_train_toy.shape, X_test_toy.shape
```

Out[51]:

```
((80, 2), (20, 2))
```

In [52]:

```

1 # join the train and test sets back together
2 XX = np.vstack((X_train_toy, X_test_toy))
3 XX.shape

```

Out[52]:

```
(100, 2)
```

In [53]:

```

1 scaler = StandardScaler()
2 scaler.fit(XX)
3 XX_scaled = scaler.transform(XX)
4 XX_train = XX_scaled[:80]
5 XX_test = XX_scaled[80:]

```

In [54]:

```

1 knn = KNeighborsClassifier()
2 knn.fit(XX_train, y_train_toy)
3 print(f"Training score: {knn.score(XX_train, y_train_toy):.2f}") # Mis
4 print(f"Test score: {knn.score(XX_test, y_test_toy):.2f}") # Misleading

```

Training score: 0.75

Test score: 0.55

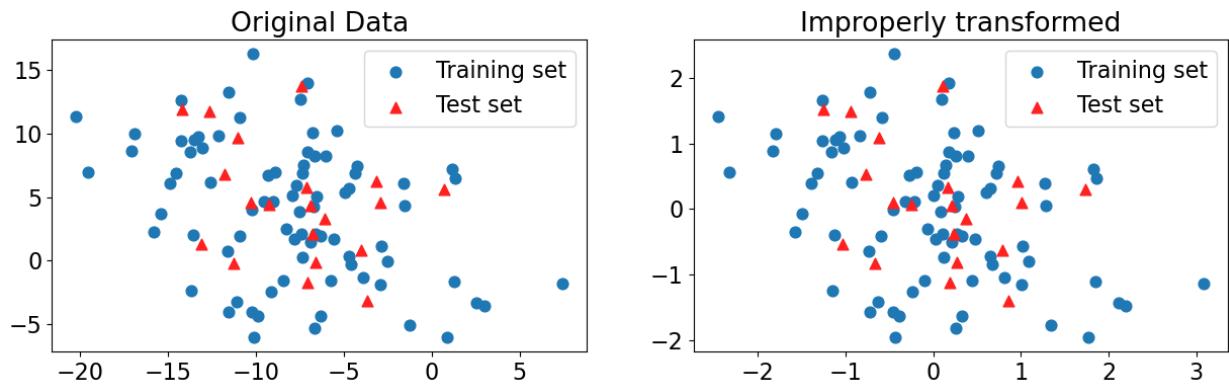
- Is anything wrong in methodology 2? If yes, what is it?

In [55]:

```

1 plot_original_scaled(
2     X_train_toy,
3     X_test_toy,
4     XX_train,
5     XX_test,
6     title_transformed="Improperly transformed",
7 )

```



Not a big difference in the transformed data but if the test set is large it might influence the mean and standard deviation significantly. **Importantly, this breaks the golden rule!**

Methodology 3: Cross validation with already preprocessed data (for class discussion)

In [56]:

```

1 knn = KNeighborsClassifier()
2
3 scaler = StandardScaler()
4 scaler.fit(X_train_toy)
5 X_train_scaled = scaler.transform(X_train_toy)
6 X_test_scaled = scaler.transform(X_test_toy)
7 scores = cross_validate(knn, X_train_scaled, y_train_toy, return_train_
8 pd.DataFrame(scores)

```

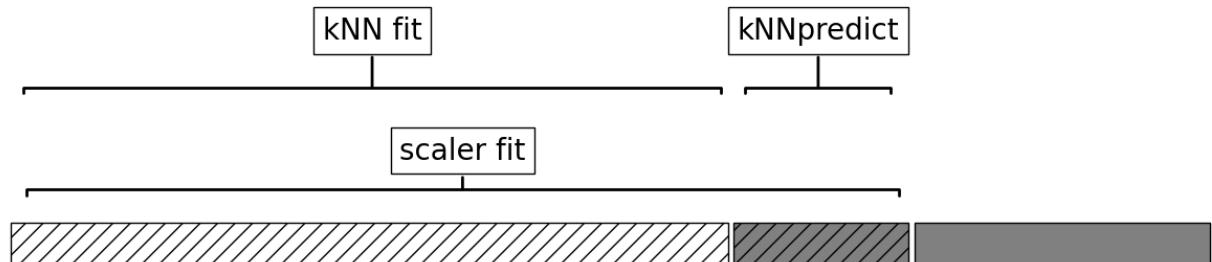
Out[56]:

	fit_time	score_time	test_score	train_score
0	0.000557	0.001731	0.6875	0.671875
1	0.000536	0.002388	0.7500	0.671875
2	0.000427	0.001202	0.6875	0.734375
3	0.000385	0.001151	0.6250	0.750000
4	0.000360	0.001179	0.5000	0.687500

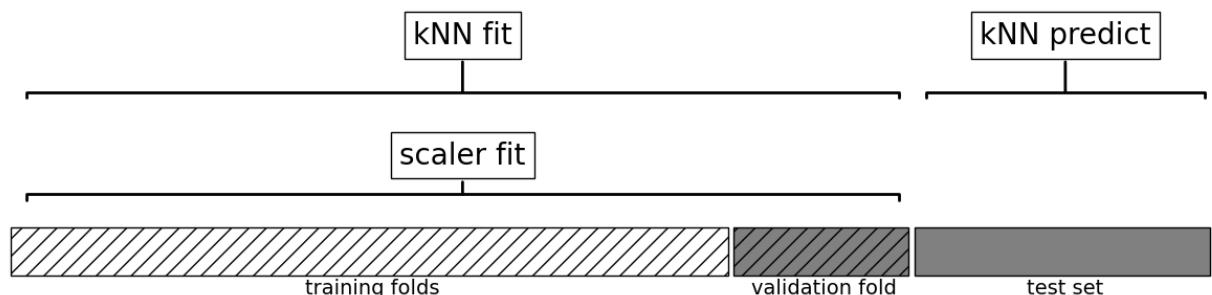
- Is there anything wrong in methodology 3? Are we breaking the golden rule here?

```
In [57]: 1 plot_improper_processing("kNN")
```

Cross validation



Test set prediction



```
In [ ]: 1 plot_proper_processing("kNN")
```

Feature transformations and the golden rule [\[video \(\)\]](#)

How to carry out cross-validation?

- Last week we saw that cross validation is a better way to get a realistic assessment of the model.
- Let's try cross-validation with transformed data.

In [58]:

```

1 knn = KNeighborsRegressor()
2
3 scaler = StandardScaler()
4 scaler.fit(X_train_imp)
5 X_train_scaled = scaler.transform(X_train_imp)
6 X_test_scaled = scaler.transform(X_test_imp)
7 scores = cross_validate(knn, X_train_scaled, y_train, return_train_score=True)
8 pd.DataFrame(scores)

```

Out[58]:

	fit_time	score_time	test_score	train_score
0	0.012000	0.262788	0.710905	0.803734
1	0.008120	0.227898	0.706893	0.803212
2	0.008050	0.243732	0.711039	0.803030
3	0.008205	0.246907	0.695769	0.806275
4	0.007997	0.181020	0.697941	0.805146

- Do you see any problem here?
- Are we applying `fit_transform` on train portion and `transform` on validation portion in each fold?
 - Here you might be allowing information from the validation set to **leak** into the training step.

- You need to apply the **SAME** preprocessing steps to train/validation.
- With many different transformations and cross validation the code gets unwieldy very quickly.
- Likely to make mistakes and "leak" information.

- In these examples our test accuracies look fine, but our methodology is flawed.
- Implications can be significant in practice!

Pipelines

Can we do this in a more elegant and organized way?

- YES!! Using [`scikit-learn Pipeline`](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html).
- [`scikit-learn Pipeline`](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html) allows you to define a "pipeline" of transformers with a final estimator.

Let's combine the preprocessing and model with pipeline

In [59]:

```

1 ### Simple example of a pipeline
2 from sklearn.pipeline import Pipeline
3
4 pipe = Pipeline(
5     steps=[
6         ("imputer", SimpleImputer(strategy="median")),
7         ("scaler", StandardScaler()),
8         ("regressor", KNeighborsRegressor()),
9     ]
10 )

```

- Syntax: pass in a list of steps.
- The last step should be a **model/classifier/regressor**.
- All the earlier steps should be **transformers**.

Alternative and more compact syntax: `make_pipeline`

- Shorthand for `Pipeline` constructor
- Does not permit naming steps
- Instead the names of steps are set to lowercase of their types automatically; `StandardScaler()` would be named as `standardscaler`

In [60]:

```

1 from sklearn.pipeline import make_pipeline
2
3 pipe = make_pipeline(
4     SimpleImputer(strategy="median"), StandardScaler(), KNeighborsRegr
5 )

```

In [61]:

```
1 pipe.fit(X_train, y_train)
```

Out[61]:

```
Pipeline(steps=[('simpleimputer', SimpleImputer(strategy='median')),
               ('standardscaler', StandardScaler()),
               ('kneighborsregressor', KNeighborsRegressor())])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

- Note that we are passing `X_train` and **not** the imputed or scaled data here.

When you call `fit` on the pipeline, it carries out the following steps:

- Fit `SimpleImputer` on `X_train`
- Transform `X_train` using the fit `SimpleImputer` to create `X_train_imp`
- Fit `StandardScaler` on `X_train_imp`
- Transform `X_train_imp` using the fit `StandardScaler` to create `X_train_imp_scaled`

- Fit the model (`KNeighborsRegressor` in our case) on `X_train_imp_scaled`

```
In [62]: 1 pipe.predict(X_train)
```

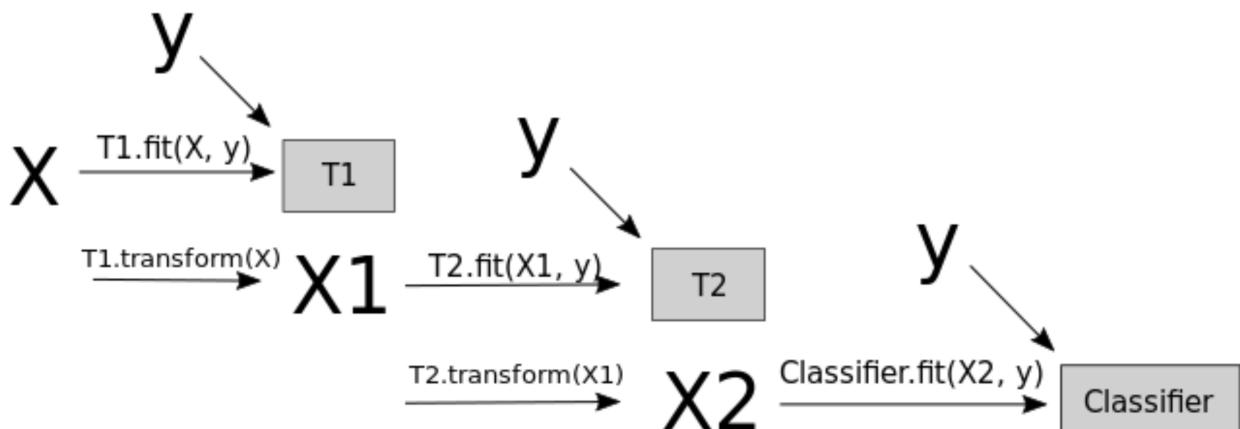
```
Out[62]: array([122460., 115220., 216940., ..., 240420., 254500., 60420.])
```

Note that we are passing original data to `predict` as well. This time the pipeline is carrying out following steps:

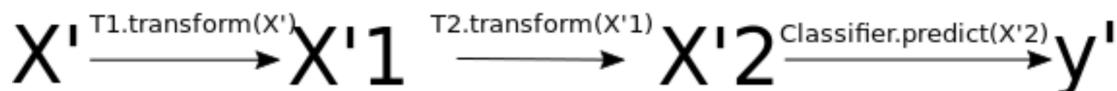
- Transform `X_train` using the fit `SimpleImputer` to create `X_train_imp`
- Transform `X_train_imp` using the fit `StandardScaler` to create `X_train_imp_scaled`
- Predict using the fit model (`KNeighborsRegressor` in our case) on `X_train_imp_scaled`.



`pipe.fit(X, y)`



`pipe.predict(X')`



Source (<https://amueller.github.io/COMS4995-s20/slides/aml-04-preprocessing/#18>)

Let's try cross-validation with our pipeline

```
In [63]: 1 results_dict["imp + scaling + knn"] = mean_std_cross_val_scores(
2     pipe, X_train, y_train, return_train_score=True
3 )
4 pd.DataFrame(results_dict).T
```

Out[63]:

	fit_time	score_time	test_score	train_score
dummy	0.002 (+/- 0.001)	0.001 (+/- 0.000)	-0.055 (+/- 0.012)	-0.055 (+/- 0.001)
imp + scaling + knn	0.028 (+/- 0.003)	0.254 (+/- 0.032)	0.706 (+/- 0.006)	0.806 (+/- 0.005)

Using a Pipeline takes care of applying the `fit_transform` on the train portion and only `transform` on the validation portion in each fold.

Break (5 min)



Categorical features [[video](#)
<https://youtu.be/2mJ9rAhMMIO>]

- Recall that we had dropped the categorical feature `ocean_proximity` feature from the dataframe. But it could potentially be a useful feature in this task.
- Let's create our `X_train` and `X_test` again by keeping the feature in the data.

In [64]: 1 `test_df.head()`

Out[64]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
19121	-122.64	38.24	40.0	1974.0	410.0	1039.0	398.0
20019	-119.05	36.09	9.0	3297.0	568.0	1749.0	568.0
15104	-116.98	32.85	12.0	3570.0	713.0	3321.0	666.0
3720	-118.42	34.20	27.0	3201.0	970.0	3403.0	948.0
8938	-118.47	34.01	41.0	2704.0	557.0	1047.0	478.0

In [66]: 1 `X_train = train_df.drop(columns=["median_house_value"])`
2 `y_train = train_df["median_house_value"]`
3
4 `X_test = test_df.drop(columns=["median_house_value"])`
5 `y_test = test_df["median_house_value"]`

- Let's try to build a `KNeighorRegressor` on this data using our pipeline

```
In [67]: 1 pipe.fit(X_train, X_train)
```

```

-----
--> 1     pipe.fit(X_train, X_train)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/pipeline.py:378, in Pipeline.fit(self, X, y, **fit_params)
    352     """Fit the model.
    353
    354 Fit all the transformers one after the other and transform the
    (...):
    375     Pipeline with fitted steps.
    376 """
    377 fit_params_steps = self._check_fit_params(**fit_params)
--> 378 Xt = self._fit(X, y, **fit_params_steps)
    379 with _print_elapsed_time("Pipeline", self._log_message(len(self.steps) - 1)):
    380     if self._final_estimator != "passthrough":

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/pipeline.py:336, in Pipeline._fit(self, X, y, **fit_params_steps)
    334     cloned_transformer = clone(transformer)
    335 # Fit or load from cache the current transformer
--> 336 X, fitted_transformer = fit_transform_one_cached(
    337     cloned_transformer,
    338     X,
    339     y,
    340     None,
    341     message_cliname="Pipeline",
    342     message=self._log_message(step_idx),
    343     **fit_params_steps[name],
    344 )
    345 # Replace the transformer of the step with the fitted
    346 # transformer. This is necessary when loading the transformer
    347 # from the cache.
    348 self.steps[step_idx] = (name, fitted_transformer)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/memorize.py:349, in NotMemorizedFunc.__call__(self, *args, **kwargs)
    348 def __call__(self, *args, **kwargs):
--> 349     return self.func(*args, **kwargs)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/pipeline.py:870, in _fit_transform_one(transformer, X, y, weight, message_cliname, message, **fit_params)
    868 with _print_elapsed_time(message_cliname, message):
    869     if hasattr(transformer, "fit_transform"):
--> 870         res = transformer.fit_transform(X, y, **fit_params)
    871     else:
    872         res = transformer.fit(X, y, **fit_params).transform(X)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/base.py:870, in TransformerMixin.fit_transform(self, X, y, **fit_params)
    867     return self.fit(X, **fit_params).transform(X)
    868 else:

```

```

869      # fit method of arity 2 (supervised transformation)
--> 870      return self.fit(X, y, **fit_params).transform(X)

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/imput
e/_base.py:364, in SimpleImputer.fit(self, X, y)
    355 if self.verbose != "deprecated":
    356     warnings.warn(
    357         "The 'verbose' parameter was deprecated in version "
    358         "'1.1' and will be removed in 1.3. A warning will "
(...):
    361         FutureWarning,
    362     )
--> 364 X = self._validate_input(X, in_fit=True)
    366 # default fill_value is 0 for numerical input and "missing_value"
    367 # otherwise
    368 if self.fill_value is None:

File ~/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/imput
e/_base.py:317, in SimpleImputer._validate_input(self, X, in_fit)
    311 if "could not convert" in str(ve):
    312     new_ve = ValueError(
    313         "Cannot use {} strategy with non-numeric data:\n{}".forma
t(
    314             self.strategy, ve
    315         )
    316     )
--> 317     raise new_ve from None
    318 else:
    319     raise ve

ValueError: Cannot use median strategy with non-numeric data:
could not convert string to float: 'INLAND'

```

- This failed because we have non-numeric data.
- Imagine how k -NN would calculate distances when you have non-numeric features.

Can we use this feature in the model?

- In scikit-learn, most algorithms require numeric inputs.
- Decision trees could theoretically work with categorical features.
 - However, the sklearn implementation does not support this.

What are the options?

- Drop the column (not recommended)
 - If you know that the column is not relevant to the target in any way you may drop it.
 - We can transform categorical features to numeric ones so that we can use them in the model.
 - [Ordinal encoding \(<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html>\)](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html)
- (occasionally recommended)

- One-hot encoding (recommended in most cases) (this lecture)

```
In [68]: 1 x_toy = pd.DataFrame(
2     {
3         "language": [
4             "English",
5             "Vietnamese",
6             "English",
7             "Mandarin",
8             "English",
9             "English",
10            "Mandarin",
11            "English",
12            "Vietnamese",
13            "Mandarin",
14            "French",
15            "Spanish",
16            "Mandarin",
17            "Hindi",
18        ]
19    }
20 )
21 x_toy
```

Out[68]:

	language
0	English
1	Vietnamese
2	English
3	Mandarin
4	English
5	English
6	Mandarin
7	English
8	Vietnamese
9	Mandarin
10	French
11	Spanish
12	Mandarin
13	Hindi

Ordinal encoding (occasionally recommended)

- Here we simply assign an integer to each of our unique categorical labels.
- We can use sklearn's [OrdinalEncoder](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html) (<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html>).

In [69]:

```

1 from sklearn.preprocessing import OrdinalEncoder
2
3 enc = OrdinalEncoder()
4 enc.fit(X_toy)
5 X_toy_ord = enc.transform(X_toy)
6 df = pd.DataFrame(
7     data=X_toy_ord,
8     columns=["language_enc"],
9     index=X_toy.index,
10 )
11 pd.concat([X_toy, df], axis=1)

```

Out[69]:

	language	language_enc
0	English	0.0
1	Vietnamese	5.0
2	English	0.0
3	Mandarin	3.0
4	English	0.0
5	English	0.0
6	Mandarin	3.0
7	English	0.0
8	Vietnamese	5.0
9	Mandarin	3.0
10	French	1.0
11	Spanish	4.0
12	Mandarin	3.0
13	Hindi	2.0

What's the problem with this approach?

- We have imposed ordinality on the categorical data.
- For example, imagine when you are calculating distances. Is it fair to say that French and Hindi are closer than French and Spanish?
- In general, label encoding is useful if there is ordinality in your data and capturing it is important for your problem, e.g., [cold, warm, hot].

One-hot encoding (OHE)

- Create new binary columns to represent our categories.
- If we have c categories in our column.
 - We create c new binary columns to represent those categories.
- Example: Imagine a language column which has the information on whether you
- We can use sklearn's [OneHotEncoder](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html) (<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html>) to do so.

One-hot encoding is called one-hot because only one of the newly created features is 1 for each data point.

In [70]:

```

1 from sklearn.preprocessing import OneHotEncoder
2
3 enc = OneHotEncoder(handle_unknown="ignore", sparse=False)
4 enc.fit(X_toy)
5 X_toy_ohe = enc.transform(X_toy)
6 pd.DataFrame(
7     data=X_toy_ohe,
8     columns=enc.get_feature_names_out(["language"]),
9     index=X_toy.index,
10 )

```

Out[70]:

	language_English	language_French	language_Hindi	language_Mandarin	language_Spanish	lang
0	1.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	1.0	0.0
4	1.0	0.0	0.0	0.0	0.0	0.0
5	1.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	1.0	0.0
7	1.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	1.0	0.0
10	0.0	1.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0	0.0	1.0
12	0.0	0.0	0.0	0.0	1.0	0.0
13	0.0	0.0	1.0	0.0	0.0	0.0

Let's do it on our housing data

In [71]:

```

1 ohe = OneHotEncoder(sparse=False, dtype="int")
2 ohe.fit(X_train[["ocean_proximity"]])
3 X_imp_ohe_train = ohe.transform(X_train[["ocean_proximity"]])

```

- We can look at the new features created using `categories_` attribute

```
In [72]: 1 ohe.categories_
```

```
Out[72]: [array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

```
In [73]: 1 transformed_ohe = pd.DataFrame(
2     data=X_imp_ohe_train,
3     columns=ohe.get_feature_names_out(["ocean_proximity"]),
4     index=X_train.index,
5 )
6 transformed_ohe
```

	ocean_proximity_<1H OCEAN	ocean_proximity_INLAND	ocean_proximity_ISLAND	ocean_proximity_NEA B/
6051	0		1	0
20113	0		1	0
14289	0		0	0
13665	0		1	0
14471	0		0	0
...
7763	1		0	0
15377	1		0	0
17730	1		0	0
15725	0		0	0
19966	0		1	0

18576 rows × 5 columns

One-hot encoded variables are also referred to as **dummy variables**. You will often see people using [get_dummies method of pandas](#) (https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html) to convert categorical variables into dummy variables. That said, using `sklearn`'s `OneHotEncoder` has the advantage of making it easy to treat training and test set in a consistent way.

?? Questions for class discussion

True/False: Pipelines and one-hot encoding

1. You can "glue" together imputation and scaling of numeric features and `scikit-learn` classifier object within a single pipeline.

2. You can "glue" together scaling of numeric features, one-hot encoding of categorical features, and `scikit-learn` classifier object within a single pipeline.
3. Pipelines will `fit` and `transform` on the training fold and only `transform` on the validation fold during cross-validation.
4. What's the better encoding for weather labels such as "sunny", "overcast" and "rainy"?

1. True
2. Not yet (we'll find a way)
3. True
4. One-hot

What did we learn today?

- Motivation for preprocessing
- Common preprocessing steps
 - Imputation
 - Scaling
 - One-hot encoding
- Golden rule in the context of preprocessing
- Building simple supervised machine learning pipelines using `sklearn.pipeline.make_pipeline`.

Problem: Different transformations on different columns

- How do we put this together with other columns in the data before fitting the regressor?
- Before we fit our regressor, we want to apply different transformations on different columns
 - Numeric columns
 - imputation
 - scaling
 - Categorical columns
 - imputation
 - one-hot encoding

Coming up: `sklearn`'s [ColumnTransformer](https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html) (<https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html>)!!

In []:

1

CPSC 330

Applied Machine Learning

Lecture 6: sklearn ColumnTransformer and Text Features

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

```
In [1]: 1 import os
2 import sys
3
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import pandas as pd
7 from IPython.display import HTML
8
9 sys.path.append("../code/.")
10 from plotting_functions import *
11 from utils import *
12
13 pd.set_option("display.max_colwidth", 200)
14
15 from sklearn.compose import ColumnTransformer, make_column_transformer
16 from sklearn.dummy import DummyClassifier, DummyRegressor
17 from sklearn.impute import SimpleImputer
18 from sklearn.model_selection import cross_val_score, cross_validate, train_test_split
19 from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
20 from sklearn.pipeline import Pipeline, make_pipeline
21 from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
22 from sklearn.svm import SVC
23 from sklearn.tree import DecisionTreeClassifier
```

Learning outcomes

From this lecture, you will be able to

- use `ColumnTransformer` to build all our transformations together into one object and use it with `sklearn` pipelines;
- define `ColumnTransformer` where transformers contain more than one steps;
- explain `handle_unknown="ignore"` hyperparameter of `scikit-learn`'s `OneHotEncoder` ;
- explain `drop="if_binary"` argument of `OneHotEncoder` ;
- identify when it's appropriate to apply ordinal encoding vs one-hot encoding;
- explain strategies to deal with categorical variables with too many categories;
- explain why text data needs a different treatment than categorical variables;
- use `scikit-learn`'s `CountVectorizer` to encode text data;
- explain different hyperparameters of `CountVectorizer` .

sklearn's `ColumnTransformer` (<https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html>)

- In most applications, some features are categorical, some are continuous, some are binary, and some are ordinal.
- When we want to develop supervised machine learning pipelines on real-world datasets, very often we want to apply different transformation on different columns.
- Enter `sklearn`'s `ColumnTransformer` !!

- Let's look at a toy example:

In [2]:

```

1 df = pd.read_csv("../data/quiz2-grade-toy-col-transformer.csv")
2 df

```

Out[2]:

	enjoy_course	ml_experience	major	class_attendance	university_years	lab1	lab2	lab3	I
0	yes	1	Computer Science	Excellent	3	92	93.0	84	
1	yes	1	Mechanical Engineering	Average	2	94	90.0	80	
2	yes	0	Mathematics	Poor	3	78	85.0	83	
3	no	0	Mathematics	Excellent	3	91	NaN	92	
4	yes	0	Psychology	Good	4	77	83.0	90	
5	no	1	Economics	Good	5	70	73.0	68	
6	yes	1	Computer Science	Excellent	4	80	88.0	89	
7	no	0	Mechanical Engineering	Poor	3	95	93.0	69	
8	no	0	Linguistics	Average	2	97	90.0	94	
9	yes	1	Mathematics	Average	4	95	82.0	94	
10	yes	0	Psychology	Good	3	98	86.0	95	
11	yes	1	Physics	Average	1	95	88.0	93	
12	yes	1	Physics	Excellent	2	98	96.0	96	
13	yes	0	Mechanical Engineering	Excellent	4	95	94.0	96	
14	no	0	Mathematics	Poor	3	95	90.0	93	
15	no	1	Computer Science	Good	3	92	85.0	67	
16	yes	0	Computer Science	Average	5	75	91.0	93	
17	yes	1	Economics	Average	3	86	89.0	65	
18	no	1	Biology	Good	2	91	NaN	90	
19	no	0	Psychology	Poor	2	77	94.0	87	
20	yes	1	Linguistics	Excellent	4	96	92.0	92	

In [3]:

```
1 df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   enjoy_course     21 non-null      object  
 1   ml_experience    21 non-null      int64   
 2   major             21 non-null      object  
 3   class_attendance 21 non-null      object  
 4   university_years  21 non-null      int64   
 5   lab1              21 non-null      int64   
 6   lab2              19 non-null      float64 
 7   lab3              21 non-null      int64   
 8   lab4              21 non-null      int64   
 9   quiz1             21 non-null      int64   
 10  quiz2             21 non-null      object  
dtypes: float64(1), int64(6), object(4)
memory usage: 1.9+ KB
```

Transformations on the toy data

In [4]:

```
1 df.head()
```

Out[4]:

	enjoy_course	ml_experience	major	class_attendance	university_years	lab1	lab2	lab3	la
0	yes	1	Computer Science	Excellent		3	92	93.0	84
1	yes	1	Mechanical Engineering	Average		2	94	90.0	80
2	yes	0	Mathematics	Poor		3	78	85.0	83
3	no	0	Mathematics	Excellent		3	91	NaN	92
4	yes	0	Psychology	Good		4	77	83.0	90

- Scaling on numeric features
- One-hot encoding on the categorical feature `major` and binary feature `enjoy_course`
- Ordinal encoding on the ordinal feature `class_attendance`
- Imputation on the `lab2` feature
- None on the `ml_experience` feature

ColumnTransformer example

Data

```
In [5]: 1 X = df.drop(columns=["quiz2"])
2 y = df["quiz2"]
3 X.columns
```

```
Out[5]: Index(['enjoy_course', 'ml_experience', 'major', 'class_attendance',
   'university_years', 'lab1', 'lab2', 'lab3', 'lab4', 'quiz1'],
  dtype='object')
```

Identify the transformations we want to apply

```
In [9]: 1 X.head()
```

```
Out[9]:   enjoy_course  ml_experience  major  class_attendance  university_years  lab1  lab2  lab3  la
0        yes            1  Computer Science  Excellent           3    92  93.0   84
1        yes            1  Mechanical Engineering  Average           2    94  90.0   80
2        yes            0  Mathematics      Poor           3    78  85.0   83
3        no             0  Mathematics      Excellent         3    91  NaN    92
4        yes            0  Psychology      Good           4    77  83.0   90
```

```
In [10]: 1 numeric_feats = ["university_years", "lab1", "lab3", "lab4", "quiz1"]
2 categorical_feats = ["major"] # apply one-hot encoding
3 passthrough_feats = ["ml_experience"] # do not apply any transformation
4 drop_feats = [
5     "lab2",
6     "class_attendance",
7     "enjoy_course",
8 ] # do not include these features in modeling
```

For simplicity, let's only focus on scaling and one-hot encoding first.

Create a column transformer

- Each transformation is specified by a name, a transformer object, and the columns this transformer should be applied to.

```
In [7]: 1 from sklearn.compose import ColumnTransformer
```

In [11]:

```

1 ct = ColumnTransformer(
2     [
3         ("scaling", StandardScaler(), numeric_feats),
4         ("onehot", OneHotEncoder(sparse=False), categorical_feats),
5     ]
6 )

```

Convenient `make_column_transformer` syntax

- Similar to `make_pipeline` syntax, there is convenient `make_column_transformer` syntax.
- The syntax automatically names each step based on its class.
- We'll be mostly using this syntax.

In [13]:

```

1 from sklearn.compose import make_column_transformer
2
3 ct = make_column_transformer(
4     (StandardScaler(), numeric_feats), # scaling on numeric features
5     (OneHotEncoder(), categorical_feats), # OHE on categorical feature
6     ("passthrough", passthrough_feats), # no transformations on the bi
7     ("drop", drop_feats), # drop the drop features
8 )

```

In [14]:

```
1 ct
```

Out[14]:

```
ColumnTransformer(transformers=[('standardscaler', StandardScaler(),
                               ['university_years', 'lab1', 'lab3', 'la
b4',
                                'quiz1']),
                               ('onehotencoder', OneHotEncoder(), ['maj
o
r']),
                               ('passthrough', 'passthrough',
                                ['ml_experience']),
                               ('drop', 'drop',
                                ['lab2', 'class_attendance', 'enjoy_cour
se'])])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [15]:

```
1 transformed = ct.fit_transform(X)
```

- When we `fit_transform`, each transformer is applied to the specified columns and the result of the transformations are concatenated horizontally.
- A big advantage here is that we build all our transformations together into one object, and that way we're sure we do the same operations to all splits of the data.
- Otherwise we might, for example, do the OHE on both train and test but forget to scale the test data.

Let's examine the transformed data

In [16]: 1 `type(transformed[:2])`

Out[16]: `numpy.ndarray`

In [17]: 1 transformed

```
Out[17]: array([[-0.09345386,  0.3589134 , -0.21733442,  0.36269995,  0.84002795,
       0.         ,  1.         ,  0.         ,  0.         ,  0.         ,
       0.         ,  0.         ,  0.         ,  1.         ],
      [-1.07471942,  0.59082668, -0.61420598, -0.85597188,  0.71219761,
       0.         ,  0.         ,  0.         ,  0.         ,  0.         ,
       1.         ,  0.         ,  0.         ,  1.         ],
      [-0.09345386, -1.26447953, -0.31655231, -1.31297381, -0.69393613,
       0.         ,  0.         ,  0.         ,  0.         ,  1.         ,
       0.         ,  0.         ,  0.         ,  0.         ],
      [-0.09345386,  0.24295676,  0.57640869,  0.36269995,  0.45653693,
       0.         ,  0.         ,  0.         ,  0.         ,  1.         ,
       0.         ,  0.         ,  0.         ,  0.         ],
      [ 0.8878117 , -1.38043616,  0.37797291,  0.51503393, -0.05478443,
       0.         ,  0.         ,  0.         ,  0.         ,  0.         ,
       0.         ,  0.         ,  1.         ,  0.         ],
      [ 1.86907725, -2.19213263, -1.80482065, -2.22697768, -1.84440919,
       0.         ,  0.         ,  1.         ,  0.         ,  0.         ,
       0.         ,  0.         ,  0.         ,  1.         ],
      [ 0.8878117 , -1.03256625,  0.27875502, -0.09430199,  0.71219761,
       0.         ,  1.         ,  0.         ,  0.         ,  0.         ,
       0.         ,  0.         ,  0.         ,  1.         ],
      [-0.09345386,  0.70678332, -1.70560276, -1.46530779, -1.33308783,
       0.         ,  0.         ,  0.         ,  0.         ,  0.         ,
       1.         ,  0.         ,  0.         ,  0.         ],
      [-1.07471942,  0.93869659,  0.77484447, -1.00830586, -0.69393613,
       0.         ,  0.         ,  0.         ,  1.         ,  0.         ,
       0.         ,  0.         ,  0.         ,  0.         ],
      [ 0.8878117 ,  0.70678332,  0.77484447,  0.81970188, -0.05478443,
       0.         ,  0.         ,  0.         ,  0.         ,  1.         ,
       0.         ,  0.         ,  0.         ,  1.         ],
      [-0.09345386,  1.05465323,  0.87406235,  0.97203586, -0.94959681,
       0.         ,  0.         ,  0.         ,  0.         ,  0.         ,
       0.         ,  0.         ,  1.         ,  0.         ],
      [-2.05598498,  0.70678332,  0.67562658,  0.51503393, -0.05478443,
       0.         ,  0.         ,  0.         ,  0.         ,  0.         ,
       0.         ,  1.         ,  0.         ,  1.         ],
      [-1.07471942,  1.05465323,  0.97328024,  1.58137177,  1.86267067,
       0.         ,  0.         ,  0.         ,  0.         ,  0.         ,
       0.         ,  1.         ,  0.         ,  1.         ],
      [ 0.8878117 ,  0.70678332,  0.97328024,  0.97203586,  1.86267067,
       0.         ,  0.         ,  0.         ,  0.         ,  0.         ,
       1.         ,  0.         ,  0.         ,  0.         ],
      [-0.09345386,  0.70678332,  0.67562658,  0.97203586, -1.97223953,
       0.         ,  0.         ,  0.         ,  0.         ,  1.         ,
       0.         ,  0.         ,  0.         ,  0.         ],
      [-0.09345386,  0.3589134 , -1.90403853,  0.81970188,  0.84002795,
       0.         ,  1.         ,  0.         ,  0.         ,  0.         ,
       0.         ,  0.         ,  0.         ,  1.         ],
      [ 1.86907725, -1.61234944,  0.67562658, -0.39896994, -0.05478443,
       0.         ,  1.         ,  0.         ,  0.         ,  0.         ,
       0.         ,  0.         ,  0.         ,  0.         ],
      [-0.09345386, -0.33682642, -2.10247431, -0.39896994,  0.20087625,
       0.         ,  0.         ,  1.         ,  0.         ,  0.         ,
       0.         ,  0.         ,  0.         ,  1.         ],
      [-1.07471942,  0.24295676,  0.37797291, -0.09430199, -0.43827545,
       1.         ,  0.         ,  0.         ,  0.         ,  0.         ,
       0.         ,  0.         ,  0.         ,  1.         ]],
```

```
[ -1.07471942, -1.38043616,  0.08031924, -1.16063983,  0.45653693,
  0.          ,  0.          ,  0.          ,  0.          ,  0.          ,
  0.          ,  0.          ,  1.          ,  0.          ],
 [ 0.8878117 ,  0.82273995,  0.57640869,  1.12436984,  0.20087625,
  0.          ,  0.          ,  0.          ,  1.          ,  0.          ,
  0.          ,  0.          ,  0.          ,  1.          ]])
```

Note that the returned object is not a dataframe. So there are no column names.

Viewing the transformed data as a dataframe

- How can we view our transformed data as a dataframe?
- We are adding more columns.
- So the original columns won't directly map to the transformed data.
- Let's create column names for the transformed data.

```
In [18]: 1 column_names = (
2     numeric_feats
3     + ct.named_transformers_[ "onehotencoder" ].get_feature_names_out().t
4     + passthrough_feats
5 )
6 column_names
```

```
Out[18]: ['university_years',
 'lab1',
 'lab3',
 'lab4',
 'quiz1',
 'major_Biology',
 'major_Computer Science',
 'major_Economics',
 'major_Linguistics',
 'major_Mathematics',
 'major_Mechanical Engineering',
 'major_Physics',
 'major_Psychology',
 'ml_experience']
```

```
In [19]: 1 ct.named_transformers_
```

```
Out[19]: {'standardscaler': StandardScaler(),
 'onehotencoder': OneHotEncoder(),
 'passthrough': 'passthrough',
 'drop': 'drop'}
```

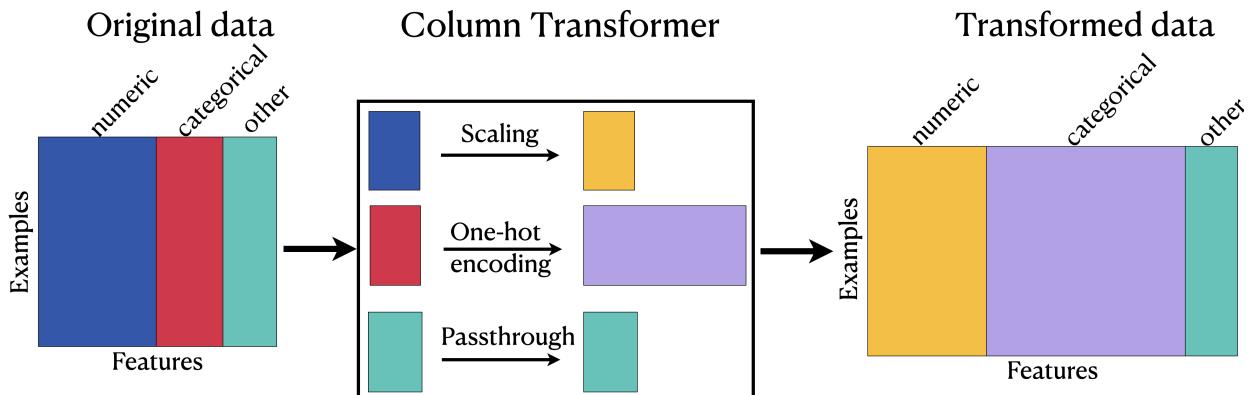
Note that the order of the columns in the transformed data depends upon the order of the features we pass to the `ColumnTransformer` and can be different than the order of the features in the original dataframe.

In [20]: 1 pd.DataFrame(transformed, columns=column_names)

Out[20]:

	university_years	lab1	lab3	lab4	quiz1	major_Biology	major_Computer Science	ma
0	-0.093454	0.358913	-0.217334	0.362700	0.840028	0.0	1.0	
1	-1.074719	0.590827	-0.614206	-0.855972	0.712198	0.0	0.0	
2	-0.093454	-1.264480	-0.316552	-1.312974	-0.693936	0.0	0.0	
3	-0.093454	0.242957	0.576409	0.362700	0.456537	0.0	0.0	
4	0.887812	-1.380436	0.377973	0.515034	-0.054784	0.0	0.0	
5	1.869077	-2.192133	-1.804821	-2.226978	-1.844409	0.0	0.0	
6	0.887812	-1.032566	0.278755	-0.094302	0.712198	0.0	1.0	
7	-0.093454	0.706783	-1.705603	-1.465308	-1.333088	0.0	0.0	
8	-1.074719	0.938697	0.774844	-1.008306	-0.693936	0.0	0.0	
9	0.887812	0.706783	0.774844	0.819702	-0.054784	0.0	0.0	
10	-0.093454	1.054653	0.874062	0.972036	-0.949597	0.0	0.0	
11	-2.055985	0.706783	0.675627	0.515034	-0.054784	0.0	0.0	
12	-1.074719	1.054653	0.973280	1.581372	1.862671	0.0	0.0	
13	0.887812	0.706783	0.973280	0.972036	1.862671	0.0	0.0	
14	-0.093454	0.706783	0.675627	0.972036	-1.972240	0.0	0.0	
15	-0.093454	0.358913	-1.904039	0.819702	0.840028	0.0	1.0	
16	1.869077	-1.612349	0.675627	-0.398970	-0.054784	0.0	1.0	
17	-0.093454	-0.336826	-2.102474	-0.398970	0.200876	0.0	0.0	
18	-1.074719	0.242957	0.377973	-0.094302	-0.438275	1.0	0.0	
19	-1.074719	-1.380436	0.080319	-1.160640	0.456537	0.0	0.0	
20	0.887812	0.822740	0.576409	1.124370	0.200876	0.0	0.0	

ColumnTransformer : Transformed data



Adapted from here. (<https://amueller.github.io/COMS4995-s20/slides/ml-04-preprocessing/#37>)

Training models with transformed data

- We can now pass the `ColumnTransformer` object as a step in a pipeline.

```
In [24]: 1 pipe = make_pipeline(ct, SVC())
2 pipe.fit(X, y)
3 pipe.predict(X)
4
5 #SVC().fit(X, y)
```

```
Out[24]: array(['A+', 'not A+', 'not A+', 'A+', 'A+', 'not A+', 'A+', 'not A+', 'not A+', 'A+', 'A+', 'A+', 'A+', 'not A+', 'not A+', 'not A+', 'not A+', 'not A+', 'not A+', 'A+', 'not A+', 'not A+', 'A+'], dtype=object)
```

?? Questions for you

True/False: `ColumnTransformer`

1. You could carry out cross-validation by passing a `ColumnTransformer` object to `cross_validate`.
2. After applying column transformer, the order of the columns in the transformed data has to be the same as the order of the columns in the original data.
3. After applying a column transformer, the transformed data is always going to be of different shape than the original data.
4. When you call `fit_transform` on a `ColumnTransformer` object, you get a numpy ndarray.

What transformations on what columns?

Consider the feature columns below.

- What transformations would you apply on each column?

colour	location	shape	water_content	weight
red	canada	NaN	84	100
yellow	mexico	long	75	120
orange	spain	NaN	90	NaN
magenta	china	round	NaN	600
purple	austria	NaN	80	115
purple	turkey	oval	78	340
green	mexico	oval	83	NaN

colour	location	shape	water_content	weight
blue	canada	round	73	535
brown	china	None	NaN	1740

More on feature transformations

Multiple transformations in a transformer

- Recall that `lab2` has missing values.

In [25]: 1 X.head(10)

	enjoy_course	ml_experience	major	class_attendance	university_years	lab1	lab2	lab3	la
0	yes	1	Computer Science	Excellent	3	92	93.0	84	
1	yes	1	Mechanical Engineering	Average	2	94	90.0	80	
2	yes	0	Mathematics	Poor	3	78	85.0	83	
3	no	0	Mathematics	Excellent	3	91	NaN	92	
4	yes	0	Psychology	Good	4	77	83.0	90	
5	no	1	Economics	Good	5	70	73.0	68	
6	yes	1	Computer Science	Excellent	4	80	88.0	89	
7	no	0	Mechanical Engineering	Poor	3	95	93.0	69	
8	no	0	Linguistics	Average	2	97	90.0	94	
9	yes	1	Mathematics	Average	4	95	82.0	94	

- So we would like to apply more than one transformations on it: imputation and scaling.
- We can treat `lab2` separately, but we can also include it into `numeric_feats` and apply both transformations on all numeric columns.

In [26]:

```

1 numeric_feats = [
2     "university_years",
3     "lab1",
4     "lab2",
5     "lab3",
6     "lab4",
7     "quiz1",
8 ] # apply scaling
9 categorical_feats = ["major"] # apply one-hot encoding
10 passthrough_feats = ["ml_experience"] # do not apply any transformation
11 drop_feats = ["class_attendance", "enjoy_course"]

```

- To apply more than one transformations we can define a pipeline inside a column transformer to chain different transformations.

In [27]:

```

1 ct = make_column_transformer(
2     (
3         make_pipeline(SimpleImputer(), StandardScaler()),
4         numeric_feats,
5     ), # scaling on numeric features
6     (OneHotEncoder(), categorical_feats), # OHE on categorical feature
7     ("passthrough", passthrough_feats), # no transformations on the bi
8     ("drop", drop_feats), # drop the drop features
9 )

```

In [28]:

```
1 X_transformed = ct.fit_transform(X)
```

In [29]:

```

1 column_names = (
2     numeric_feats
3     + ct.named_transformers_[ "onehotencoder" ].get_feature_names_out().t
4     + passthrough_feats
5 )
6 column_names

```

Out[29]:

```

['university_years',
 'lab1',
 'lab2',
 'lab3',
 'lab4',
 'quiz1',
 'major_Biology',
 'major_Computer Science',
 'major_Economics',
 'major_Linguistics',
 'major_Mathematics',
 'major_Mechanical Engineering',
 'major_Physics',
 'major_Psychology',
 'ml_experience']

```

In [30]:

```
1 pd.DataFrame(X_transformed, columns=column_names)
```

Out[30]:

	university_years	lab1	lab2	lab3	lab4	quiz1	major_Biology	major_Cor_S
0	-0.093454	0.358913	0.893260	-0.217334	0.362700	0.840028		0.0
1	-1.074719	0.590827	0.294251	-0.614206	-0.855972	0.712198		0.0
2	-0.093454	-1.264480	-0.704099	-0.316552	-1.312974	-0.693936		0.0
3	-0.093454	0.242957	0.000000	0.576409	0.362700	0.456537		0.0
4	0.887812	-1.380436	-1.103439	0.377973	0.515034	-0.054784		0.0
5	1.869077	-2.192133	-3.100139	-1.804821	-2.226978	-1.844409		0.0
6	0.887812	-1.032566	-0.105089	0.278755	-0.094302	0.712198		0.0
7	-0.093454	0.706783	0.893260	-1.705603	-1.465308	-1.333088		0.0
8	-1.074719	0.938697	0.294251	0.774844	-1.008306	-0.693936		0.0
9	0.887812	0.706783	-1.303109	0.774844	0.819702	-0.054784		0.0
10	-0.093454	1.054653	-0.504429	0.874062	0.972036	-0.949597		0.0
11	-2.055985	0.706783	-0.105089	0.675627	0.515034	-0.054784		0.0
12	-1.074719	1.054653	1.492270	0.973280	1.581372	1.862671		0.0
13	0.887812	0.706783	1.092930	0.973280	0.972036	1.862671		0.0
14	-0.093454	0.706783	0.294251	0.675627	0.972036	-1.972240		0.0
15	-0.093454	0.358913	-0.704099	-1.904039	0.819702	0.840028		0.0
16	1.869077	-1.612349	0.493921	0.675627	-0.398970	-0.054784		0.0
17	-0.093454	-0.336826	0.094581	-2.102474	-0.398970	0.200876		0.0
18	-1.074719	0.242957	0.000000	0.377973	-0.094302	-0.438275		1.0
19	-1.074719	-1.380436	1.092930	0.080319	-1.160640	0.456537		0.0
20	0.887812	0.822740	0.693590	0.576409	1.124370	0.200876		0.0

sklearn set_config

- With multiple transformations in a column transformer, it can get tricky to keep track of everything happening inside it.
- We can use `set_config` to display a diagram of this.

```
In [31]: 1 from sklearn import set_config
          2
          3 set_config(display="diagram")
```

```
In [32]: 1 ct
```

```
Out[32]: ColumnTransformer(transformers=[('pipeline',
                                         Pipeline(steps=[('simpleimputer',
                                                          SimpleImputer()),
                                                          ('standardscaler',
                                                          StandardScaler())])),
                                         ['university_years', 'lab1', 'lab2', 'la
                                          b3',
                                          'lab4', 'quiz1']),
                                         ('onehotencoder', OneHotEncoder(), ['majo
                                          r']),
                                         ('passthrough', 'passthrough',
                                         ['ml_experience']),
                                         ('drop', 'drop',
                                         ['class_attendance', 'enjoy_course']))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [33]: 1 print(ct)
```

```
ColumnTransformer(transformers=[('pipeline',
                                 Pipeline(steps=[('simpleimputer',
                                                 SimpleImputer()),
                                                 ('standardscaler',
                                                 StandardScaler())])),
                                 ['university_years', 'lab1', 'lab2', 'la
                                  b3',
                                  'lab4', 'quiz1']),
                                 ('onehotencoder', OneHotEncoder(), ['majo
                                  r']),
                                 ('passthrough', 'passthrough',
                                 ['ml_experience']),
                                 ('drop', 'drop',
                                 ['class_attendance', 'enjoy_course']))])
```

Incorporating ordinal feature `class_attendance`

- The `class_attendance` column is different than the `major` column in that there is some ordering of the values.

- Excellent > Good > Average > poor

In [34]: 1 X.head()

	enjoy_course	ml_experience	major	class_attendance	university_years	lab1	lab2	lab3	la
0	yes	1	Computer Science	Excellent		3	92	93.0	84
1	yes	1	Mechanical Engineering	Average		2	94	90.0	80
2	yes	0	Mathematics	Poor		3	78	85.0	83
3	no	0	Mathematics	Excellent		3	91	NaN	92
4	yes	0	Psychology	Good		4	77	83.0	90

Let's try applying `OrdinalEncoder` on this column.

In [36]:

```

1 X_toy = X[["class_attendance"]]
2 enc = OrdinalEncoder()
3 enc.fit(X_toy)
4 X_toy_ord = enc.transform(X_toy)
5 df = pd.DataFrame(
6     data=X_toy_ord,
7     columns=["class_attendance_enc"],
8     index=X_toy.index,
9 )

```

In [37]: 1 pd.concat([X_toy, df], axis=1).head(10)

Out[37]:

	class_attendance	class_attendance_enc
0	Excellent	1.0
1	Average	0.0
2	Poor	3.0
3	Excellent	1.0
4	Good	2.0
5	Good	2.0
6	Excellent	1.0
7	Poor	3.0
8	Average	0.0
9	Average	0.0

- What's the problem here?
 - The encoder doesn't know the order.
- We can examine unique categories manually, order them based on our intuitions, and then provide this human knowledge to the transformer.

What are the unique categories of `class_attendance` ?

In [38]: 1 `x_toy["class_attendance"].unique()`

Out[38]: `array(['Excellent', 'Average', 'Poor', 'Good'], dtype=object)`

Let's order them manually.

In [39]: 1 `class_attendance_levels = ["Poor", "Average", "Good", "Excellent"]`

Note that if you use the reverse order of the categories, it would n't matter.

Let's make sure that we have included all categories in our manual ordering.

In [40]: 1 `assert set(class_attendance_levels) == set(x_toy["class_attendance"].un`

In [41]: 1 `oe = OrdinalEncoder(categories=[class_attendance_levels], dtype=int)`
 2 `oe.fit(x_toy[["class_attendance"]])`
 3 `ca_transformed = oe.transform(x_toy[["class_attendance"]])`
 4 `df = pd.DataFrame(`
 5 `data=ca_transformed, columns=["class_attendance_enc"], index=x_toy.`
 6 `)`
 7 `print(oe.categories_)`
 8 `pd.concat([x_toy, df], axis=1).head(10)`

`[array(['Poor', 'Average', 'Good', 'Excellent'], dtype=object)]`

Out[41]: `class_attendance class_attendance_enc`

0	Excellent	3
1	Average	1
2	Poor	0
3	Excellent	3
4	Good	2
5	Good	2
6	Excellent	3
7	Poor	0
8	Average	1
9	Average	1

The encoded categories are looking better now!

More than one ordinal columns?

- We can pass the manually ordered categories when we create an `OrdinalEncoder` object as a list of lists.
- If you have more than one ordinal columns
 - manually create a list of ordered categories for each column
 - pass a list of lists to `OrdinalEncoder`, where each inner list corresponds to manually created list of ordered categories for a corresponding ordinal column.

Now let's incorporate ordinal encoding of `class_attendance` in our column transformer.

In [42]:

```

1 numeric_feats = [
2     "university_years",
3     "lab1",
4     "lab2",
5     "lab3",
6     "lab4",
7     "quiz1",
8 ] # apply scaling
9 categorical_feats = ["major"] # apply one-hot encoding
10 ordinal_feats = ["class_attendance"] # apply ordinal encoding
11 passthrough_feats = ["ml_experience"] # do not apply any transformation
12 drop_feats = ["enjoy_course"] # do not include these features

```

In [43]:

```

1 ct = make_column_transformer(
2     (
3         make_pipeline(SimpleImputer(), StandardScaler()),
4         numeric_feats,
5     ), # scaling on numeric features
6     (OneHotEncoder(), categorical_feats), # OHE on categorical feature
7     (
8         OrdinalEncoder(categories=[class_attendance_levels], dtype=int),
9         ordinal_feats,
10    ), # Ordinal encoding on ordinal features
11    ("passthrough", passthrough_feats), # no transformations on the bi
12    ("drop", drop_feats), # drop the drop features
13 )

```

```
In [44]: 1 ct
```

```
Out[44]: ColumnTransformer(transformers=[('pipeline',
                                         Pipeline(steps=[('simpleimputer',
                                                          SimpleImputer()),
                                                          ('standardscaler',
                                                          StandardScaler())])),
                                         ['university_years', 'lab1', 'lab2', 'la
                                          b3',
                                          'lab4', 'quiz1']),
                                         ('onehotencoder', OneHotEncoder(), ['majo
                                          r']),
                                         ('ordinalencoder',
                                         OrdinalEncoder(categories=[[['Poor', 'Ave
                                          rage',
                                          'Good',
                                          'Excellen
                                          t']],
                                         dtype=<class 'int'>),
                                         ['class_attendance']),
                                         ('passthrough', 'passthrough',
                                         ['ml_experience']),
                                         ('drop', 'drop', ['enjoy_course']))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [45]: 1 x_transformed = ct.fit_transform(X)
```

In [46]:

```
1 column_names = (
2     numeric_feats
3     + ct.named_transformers_[ "onehotencoder" ].get_feature_names_out().t
4     + ordinal_feats
5     + passthrough_feats
6 )
7 column_names
```

Out[46]:

```
['university_years',
 'lab1',
 'lab2',
 'lab3',
 'lab4',
 'quiz1',
 'major_Biology',
 'major_Computer Science',
 'major_Economics',
 'major_Linguistics',
 'major_Mathematics',
 'major_Mechanical Engineering',
 'major_Physics',
 'major_Psychology',
 'class_attendance',
 'ml_experience']
```

In [47]: 1 pd.DataFrame(X_transformed, columns=column_names)

Out[47]:

	university_years	lab1	lab2	lab3	lab4	quiz1	major_Biology	major_Cor_S
0	-0.093454	0.358913	0.893260	-0.217334	0.362700	0.840028		0.0
1	-1.074719	0.590827	0.294251	-0.614206	-0.855972	0.712198		0.0
2	-0.093454	-1.264480	-0.704099	-0.316552	-1.312974	-0.693936		0.0
3	-0.093454	0.242957	0.000000	0.576409	0.362700	0.456537		0.0
4	0.887812	-1.380436	-1.103439	0.377973	0.515034	-0.054784		0.0
5	1.869077	-2.192133	-3.100139	-1.804821	-2.226978	-1.844409		0.0
6	0.887812	-1.032566	-0.105089	0.278755	-0.094302	0.712198		0.0
7	-0.093454	0.706783	0.893260	-1.705603	-1.465308	-1.333088		0.0
8	-1.074719	0.938697	0.294251	0.774844	-1.008306	-0.693936		0.0
9	0.887812	0.706783	-1.303109	0.774844	0.819702	-0.054784		0.0
10	-0.093454	1.054653	-0.504429	0.874062	0.972036	-0.949597		0.0
11	-2.055985	0.706783	-0.105089	0.675627	0.515034	-0.054784		0.0
12	-1.074719	1.054653	1.492270	0.973280	1.581372	1.862671		0.0
13	0.887812	0.706783	1.092930	0.973280	0.972036	1.862671		0.0
14	-0.093454	0.706783	0.294251	0.675627	0.972036	-1.972240		0.0
15	-0.093454	0.358913	-0.704099	-1.904039	0.819702	0.840028		0.0
16	1.869077	-1.612349	0.493921	0.675627	-0.398970	-0.054784		0.0
17	-0.093454	-0.336826	0.094581	-2.102474	-0.398970	0.200876		0.0
18	-1.074719	0.242957	0.000000	0.377973	-0.094302	-0.438275		1.0
19	-1.074719	-1.380436	1.092930	0.080319	-1.160640	0.456537		0.0
20	0.887812	0.822740	0.693590	0.576409	1.124370	0.200876		0.0

Dealing with unknown categories

Let's create a pipeline with the column transformer and pass it to `cross_validate`.

In [48]: 1 pipe = make_pipeline(ct, SVC())

In [49]:

```
1 scores = cross_validate(pipe, X, y, return_train_score=True)
2 pd.DataFrame(scores)
```

```
/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklear
rn/model_selection/_validation.py:776: UserWarning: Scoring failed. The s
core on this train-test partition for these parameters will be set to na
n. Details:
Traceback (most recent call last):
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklear
n/model_selection/_validation.py", line 767, in _score
    scores = scorer(estimator, X_test, y_test)
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklear
n/metrics/_scorer.py", line 429, in _passthrough_scorer
    return estimator.score(*args, **kwargs)
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklear
n/pipeline.py", line 695, in score
    Xt = transform.transform(Xt)
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklear
n/compose/_column_transformer.py", line 763, in transform
    Xs = self._fit_transform(
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklear
n/compose/_column_transformer.py", line 621, in _fit_transform
    return Parallel(n_jobs=self.n_jobs)(
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/parallel.py", line 1051, in __call__
    while self.dispatch_one_batch(iterator):
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/parallel.py", line 864, in dispatch_one_batch
    self._dispatch(tasks)
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/parallel.py", line 782, in _dispatch
    job = self._backend.apply_async(batch, callback=cb)
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/_parallel_backends.py", line 208, in apply_async
    result = ImmediateResult(func)
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/_parallel_backends.py", line 572, in __init__
    self.results = batch()
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/parallel.py", line 263, in __call__
    return [func(*args, **kwargs)]
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/joblib/parallel.py", line 263, in <listcomp>
    return [func(*args, **kwargs)]
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/utils/fixed.py", line 117, in __call__
    return self.function(*args, **kwargs)
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklear
n/pipeline.py", line 853, in _transform_one
    res = transformer.transform(X)
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklear
n/preprocessing/_encoders.py", line 882, in transform
    X_int, X_mask = self._transform(
  File "/Users/mathias/miniconda3/envs/cpsc330/lib/python3.10/site-packages/sklear
n/preprocessing/_encoders.py", line 160, in _transform
    raise ValueError(msg)
ValueError: Found unknown categories ['Biology'] in column 0 during trans
form
```

```
warnings.warn(
```

Out[49]:

	fit_time	score_time	test_score	train_score
0	0.013106	0.006488	1.00	0.937500
1	0.010220	0.005606	1.00	0.941176
2	0.008559	0.005460	0.50	1.000000
3	0.007937	0.004804	0.75	0.941176
4	0.008821	0.024068	NaN	1.000000

- What's going on here??
- Let's look at the error message: `ValueError: Found unknown categories ['Biology'] in column 0 during transform`

In [50]: 1 X["major"].value_counts()

Out[50]:

Computer Science	4
Mathematics	4
Mechanical Engineering	3
Psychology	3
Economics	2
Linguistics	2
Physics	2
Biology	1

Name: major, dtype: int64

- There is only one instance of Biology.
- During cross-validation, this is getting put into the validation split.
- By default, `OneHotEncoder` throws an error because you might want to know about this.

Simplest fix:

- Pass `handle_unknown="ignore"` argument to `OneHotEncoder`
- It creates a row with all zeros.

```
In [51]: 1 ct = make_column_transformer(
2     (
3         make_pipeline(SimpleImputer(), StandardScaler()),
4         numeric_feats,
5     ), # scaling on numeric features
6     (
7         OneHotEncoder(handle_unknown="ignore"),
8         categorical_feats,
9     ), # OHE on categorical features
10    (
11        OrdinalEncoder(categories=[class_attendance_levels], dtype=int),
12        ordinal_feats,
13    ), # Ordinal encoding on ordinal features
14    ("passthrough", passthrough_feats), # no transformations on the bi
15    ("drop", drop_feats), # drop the drop features
16 )
```

```
In [52]: 1 ct
```

```
Out[52]: ColumnTransformer(transformers=[('pipeline',
                                         Pipeline(steps=[('simpleimputer',
                                                          SimpleImputer()),
                                                          ('standardscaler',
                                                          StandardScaler())]),
                                         ['university_years', 'lab1', 'lab2', 'la
b3',
                                          'lab4', 'quiz1']),
                                         ('onehotencoder',
                                          OneHotEncoder(handle_unknown='ignore'),
                                          ['major']),
                                         ('ordinalencoder',
                                          OrdinalEncoder(categories=[[ 'Poor', 'Ave
rage',
                                          'Good',
                                          'Excellen
t'],
                                         dtype=<class 'int'>),
                                         ['class_attendance'])),
                                         ('passthrough', 'passthrough',
                                         ['ml_experience']),
                                         ('drop', 'drop', ['enjoy_course'])]))
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [53]: 1 pipe = make_pipeline(ct, SVC())
```

In [54]:

```
1 scores = cross_validate(pipe, X, y, cv=5, return_train_score=True)
2 pd.DataFrame(scores)
```

Out[54]:

	fit_time	score_time	test_score	train_score
0	0.010619	0.006246	1.00	0.937500
1	0.011235	0.005561	1.00	0.941176
2	0.009199	0.005520	0.50	1.000000
3	0.009306	0.005152	0.75	0.941176
4	0.008154	0.004914	0.75	1.000000

- With this approach, all unknown categories will be represented with all zeros and cross-validation is running OK now.

Ask yourself the following questions when you work with categorical variables

- Do you want this behaviour?
- Are you expecting to get many unknown categories? Do you want to be able to distinguish between them?

Cases where it's OK to break the golden rule

- If it's some fix number of categories. For example, if it's something like provinces in Canada or majors taught at UBC. We know the categories in advance and this is one of the cases where it might be OK to violate the golden rule and get a list of all possible values for the categorical variable.

Categorical features with only two possible categories

- Sometimes you have features with only two possible categories.
- If we apply `OheHotEncoder` on such columns, it'll create two columns, which seems wasteful, as we could represent all information in the column in just one column with say 0's and 1's with presence of absence of one of the categories.
- You can pass `drop="if_binary"` argument to `OneHotEncoder` in order to create only one column in such scenario.

```
In [55]: 1 X[ "enjoy_course" ].head()
```

```
Out[55]: 0    yes
1    yes
2    yes
3    no
4    yes
Name: enjoy_course, dtype: object
```

```
In [56]: 1 ohe_enc = OneHotEncoder(drop="if_binary", dtype=int, sparse=False)
2 ohe_enc.fit(X[["enjoy_course"]])
3 transformed = ohe_enc.transform(X[["enjoy_course"]])
4 df = pd.DataFrame(data=transformed, columns=["enjoy_course_enc"], index
5 pd.concat([X[["enjoy_course"]], df], axis=1).head(10)
```

	enjoy_course	enjoy_course_enc
0	yes	1
1	yes	1
2	yes	1
3	no	0
4	yes	1
5	no	0
6	yes	1
7	no	0
8	no	0
9	yes	1

```
In [57]: 1 numeric_feats = [
2     "university_years",
3     "lab1",
4     "lab2",
5     "lab3",
6     "lab4",
7     "quiz1",
8 ] # apply scaling
9 categorical_feats = ["major"] # apply one-hot encoding
10 ordinal_feats = ["class_attendance"] # apply ordinal encoding
11 binary_feats = ["enjoy_course"] # apply one-hot encoding with drop="if
12 passthrough_feats = ["ml_experience"] # do not apply any transformatio
13 drop_feats = []
```

In [59]:

```
1 ct = make_column_transformer(
2     (
3         make_pipeline(SimpleImputer(), StandardScaler()),
4         numeric_feats,
5     ), # scaling on numeric features
6     (
7         OneHotEncoder(handle_unknown="ignore"),
8         categorical_feats,
9     ), # OHE on categorical features
10    (
11        OrdinalEncoder(categories=[class_attendance_levels], dtype=int),
12        ordinal_feats,
13    ), # Ordinal encoding on ordinal features
14    (
15        OneHotEncoder(drop="if_binary", dtype=int),
16        binary_feats,
17    ), # OHE on categorical features
18    ("passthrough", passthrough_feats), # no transformations on the bi
19 )
```

In [60]: 1 ct

```
Out[60]: ColumnTransformer(transformers=[('pipeline',
                                         Pipeline(steps=[('simpleimputer',
                                                          SimpleImputer()),
                                                          ('standardscaler',
                                                          StandardScaler())])),
                                         ['university_years', 'lab1', 'lab2', 'la
                                          b3',
                                          'lab4', 'quiz1']),
                                         ('onehotencoder-1',
                                         OneHotEncoder(handle_unknown='ignore'),
                                         ['major']),
                                         ('ordinalencoder',
                                         OrdinalEncoder(categories=[[ 'Poor', 'Ave
                                          rage',
                                          'Good',
                                          'Excellen
                                          t']])),
                                         dtype=<class 'int'>),
                                         ['class_attendance']),
                                         ('onehotencoder-2',
                                         OneHotEncoder(drop='if_binary',
                                         dtype=<class 'int'>),
                                         ['enjoy_course']),
                                         ('passthrough', 'passthrough',
                                         ['ml_experience'])])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [61]: 1 pipe = make_pipeline(ct, SVC())

In [62]: 1 scores = cross_validate(pipe, X, y, cv=5, return_train_score=True)
2 pd.DataFrame(scores)

Out[62]:

	fit_time	score_time	test_score	train_score
0	0.013232	0.008059	1.00	1.000000
1	0.011555	0.006521	1.00	0.941176
2	0.010359	0.006901	0.50	1.000000
3	0.010653	0.006728	1.00	0.941176
4	0.009934	0.006208	0.75	1.000000

Do not read too much into the scores, as we are running cross-validation on a very small dataset with 21 examples. The main point here is to show you how we can use `ColumnTransformer` to apply different transformations on different columns.

Break (5 min)



ColumnTransformer on the California housing dataset

```
In [63]: 1 housing_df = pd.read_csv("../data/housing.csv")
2 train_df, test_df = train_test_split(housing_df, test_size=0.1, random_
3
4 train_df.head()
```

Out[63]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	
6051	-117.75	34.04		22.0	2948.0	636.0	2600.0	602.0
20113	-119.57	37.94		17.0	346.0	130.0	51.0	20.0
14289	-117.13	32.74		46.0	3355.0	768.0	1457.0	708.0
13665	-117.31	34.02		18.0	1634.0	274.0	899.0	285.0
14471	-117.23	32.88		18.0	5566.0	1465.0	6303.0	1458.0

Some column values are mean/median but some are not.

Let's add some new features to the dataset which could help predicting the target:
`median_house_value`.

```
In [64]: 1 train_df = train_df.assign(
2     rooms_per_household=train_df["total_rooms"] / train_df["households"]
3 )
4 test_df = test_df.assign(
5     rooms_per_household=test_df["total_rooms"] / test_df["households"]
6 )
7
8 train_df = train_df.assign(
9     bedrooms_per_household=train_df["total_bedrooms"] / train_df["households"]
10)
11 test_df = test_df.assign(
12     bedrooms_per_household=test_df["total_bedrooms"] / test_df["households"]
13)
14
15 train_df = train_df.assign(
16     population_per_household=train_df["population"] / train_df["households"]
17)
18 test_df = test_df.assign(
19     population_per_household=test_df["population"] / test_df["households"]
20)
```

In [65]: 1 train_df.head()

Out[65]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
6051	-117.75	34.04	22.0	2948.0	636.0	2600.0	602.0
20113	-119.57	37.94	17.0	346.0	130.0	51.0	20.0
14289	-117.13	32.74	46.0	3355.0	768.0	1457.0	708.0
13665	-117.31	34.02	18.0	1634.0	274.0	899.0	285.0
14471	-117.23	32.88	18.0	5566.0	1465.0	6303.0	1458.0

In [66]:

```
1 # Let's keep both numeric and categorical columns in the data.
2 X_train = train_df.drop(columns=["median_house_value"])
3 y_train = train_df["median_house_value"]
4
5 X_test = test_df.drop(columns=["median_house_value"])
6 y_test = test_df["median_house_value"]
```

In [67]: 1 from sklearn.compose import ColumnTransformer, make_column_transformer

In [68]: 1 X_train.head(10)

Out[68]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households
6051	-117.75	34.04	22.0	2948.0	636.0	2600.0	602.0
20113	-119.57	37.94	17.0	346.0	130.0	51.0	20.0
14289	-117.13	32.74	46.0	3355.0	768.0	1457.0	708.0
13665	-117.31	34.02	18.0	1634.0	274.0	899.0	285.0
14471	-117.23	32.88	18.0	5566.0	1465.0	6303.0	1458.0
9730	-121.74	36.79	16.0	3841.0	620.0	1799.0	611.0
14690	-117.09	32.80	36.0	2163.0	367.0	915.0	360.0
7938	-118.11	33.86	33.0	2389.0	410.0	1229.0	393.0
18365	-122.12	37.28	21.0	349.0	64.0	149.0	56.0
10931	-117.91	33.74	25.0	4273.0	965.0	2946.0	922.0

In [69]: 1 X_train.columns

Out[69]:

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'ocean_proximity', 'rooms_per_household', 'bedrooms_per_househol
d',
       'population_per_household'],
      dtype='object')
```

In [70]:

```

1 # Identify the categorical and numeric columns
2 numeric_features = [
3     "longitude",
4     "latitude",
5     "housing_median_age",
6     "total_rooms",
7     "total_bedrooms",
8     "population",
9     "households",
10    "median_income",
11    "rooms_per_household",
12    "bedrooms_per_household",
13    "population_per_household",
14 ]
15
16 categorical_features = ["ocean_proximity"]
17 target = "median_income"

```

- Let's create a `ColumnTransformer` for our dataset.

In [71]:

```
1 X_train.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 18576 entries, 6051 to 19966
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   longitude        18576 non-null   float64
 1   latitude         18576 non-null   float64
 2   housing_median_age 18576 non-null   float64
 3   total_rooms      18576 non-null   float64
 4   total_bedrooms   18391 non-null   float64
 5   population       18576 non-null   float64
 6   households       18576 non-null   float64
 7   median_income    18576 non-null   float64
 8   ocean_proximity 18576 non-null   object  
 9   rooms_per_household 18576 non-null   float64
 10  bedrooms_per_household 18391 non-null   float64
 11  population_per_household 18576 non-null   float64
dtypes: float64(11), object(1)
memory usage: 1.8+ MB

```

In [72]:

```
1 X_train["ocean_proximity"].value_counts()
```

```

Out[72]: <1H OCEAN      8221
          INLAND       5915
          NEAR OCEAN    2389
          NEAR BAY      2046
          ISLAND        5
Name: ocean_proximity, dtype: int64

```

```
In [73]: 1 numeric_transformer = make_pipeline(SimpleImputer(strategy="median"), s
2 categorical_transformer = OneHotEncoder(handle_unknown="ignore")
3
4 preprocessor = make_column_transformer(
5     (numeric_transformer, numeric_features),
6     (categorical_transformer, categorical_features),
7 )
```

```
In [74]: 1 preprocessor
```

```
Out[74]: ColumnTransformer(transformers=[('pipeline',
                                         Pipeline(steps=[('simpleimputer',
                                                          SimpleImputer(strategy
='median'))),
                                         ('standardscaler',
                                          StandardScaler()))]),
                           ['longitude', 'latitude', 'housing_median_
age',
                            'total_rooms', 'total_bedrooms', 'popul
ation',
                            'households', 'median_income',
                            'rooms_per_household',
                            'bedrooms_per_household',
                            'population_per_household']),
                           ('onehotencoder',
                            OneHotEncoder(handle_unknown='ignore'),
                            ['ocean_proximity']))]
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [75]: 1 X_train_pp = preprocessor.fit_transform(X_train)
```

- When we `fit` the preprocessor, it calls `fit` on *all* the transformers
- When we `transform` the preprocessor, it calls `transform` on *all* the transformers.

We can get the new names of the columns that were generated by the one-hot encoding:

```
In [76]: 1 preprocessor
```

```
Out[76]: ColumnTransformer(transformers=[('pipeline',
                                         Pipeline(steps=[('simpleimputer',
                                                          SimpleImputer(strategy
='median')),
                                         ('standardscaler',
                                          StandardScaler()))]),
                                         ['longitude', 'latitude', 'housing_media_
n_age',
                                         'total_rooms', 'total_bedrooms', 'popul_
ation',
                                         'households', 'median_income',
                                         'rooms_per_household',
                                         'bedrooms_per_household',
                                         'population_per_household']),
                                         ('onehotencoder',
                                          OneHotEncoder(handle_unknown='ignore'),
                                          ['ocean_proximity'])])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [77]: 1 preprocessor.named_transformers_["onehotencoder"].get_feature_names_out
2     categorical_features
3 )
```

```
Out[77]: array(['ocean_proximity_<1H OCEAN', 'ocean_proximity_INLAND',
   'ocean_proximity_ISLAND', 'ocean_proximity_NEAR BAY',
   'ocean_proximity_NEAR OCEAN'], dtype=object)
```

Combining this with the numeric feature names gives us all the column names:

```
In [78]: 1 column_names = numeric_features + list(
2     preprocessor.named_transformers_[ "onehotencoder" ].get_feature_names
3         categorical_features
4     )
5 )
6 column_names
```

```
Out[78]: ['longitude',
'latitude',
'housing_median_age',
'total_rooms',
'total_bedrooms',
'population',
'households',
'median_income',
'rooms_per_household',
'bedrooms_per_household',
'population_per_household',
'ocean_proximity_<1H OCEAN',
'ocean_proximity_INLAND',
'ocean_proximity_ISLAND',
'ocean_proximity_NEAR BAY',
'ocean_proximity_NEAR OCEAN']
```

Let's visualize the preprocessed training data as a dataframe.

```
In [79]: 1 pd.DataFrame(X_train_pp, columns=column_names)
```

Out[79]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	househol
0	0.908140	-0.743917	-0.526078	0.143120	0.235339	1.026092	0.2661
1	-0.002057	1.083123	-0.923283	-1.049510	-0.969959	-1.206672	-1.2533
2	1.218207	-1.352930	1.380504	0.329670	0.549764	0.024896	0.5428
3	1.128188	-0.753286	-0.843842	-0.459154	-0.626949	-0.463877	-0.5614
4	1.168196	-1.287344	-0.843842	1.343085	2.210026	4.269688	2.5009
...
18571	0.733102	-0.804818	0.586095	-0.875337	-0.243446	-0.822136	-0.9661
18572	1.163195	-1.057793	-1.161606	0.940194	0.609314	0.882438	0.7282
18573	-1.097293	0.797355	-1.876574	0.695434	0.433046	0.881563	0.5141
18574	-1.437367	1.008167	1.221622	-0.499947	-0.484029	-0.759944	-0.4544
18575	0.242996	0.272667	-0.684960	-0.332190	-0.353018	-0.164307	-0.3969

18576 rows × 16 columns

In [80]:

```
1 results_dict = {}
2 dummy = DummyRegressor()
3 results_dict[ "dummy" ] = mean_std_cross_val_scores(
4     dummy, X_train, y_train, return_train_score=True
5 )
6 pd.DataFrame(results_dict).T
```

Out[80]:

	fit_time	score_time	test_score	train_score
dummy	0.002 (+/- 0.001)	0.000 (+/- 0.000)	-0.001 (+/- 0.001)	0.000 (+/- 0.000)

In [81]:

```
1 from sklearn.svm import SVR
2
3 knn_pipe = make_pipeline(preprocessor, KNeighborsRegressor())
```

In [82]: 1 knn_pipe

```
Out[82]: Pipeline(steps=[('columntransformer',
                         ColumnTransformer(transformers=[('pipeline',
                         Pipeline(steps=[('simpleimputer',
                                         SimpleImputer(strategy='median'))),
                         ('standardscaler',
                                         StandardScaler())),
                         ['longitude', 'latitude',
                          'housing_median_age',
                          'total_rooms',
                          'total_bedrooms',
                          'population', 'households',
                          'median_income',
                          'rooms_per_household',
                          'bedrooms_per_household'],
                         'population_per_household'],
                         ('onehotencoder',
                                         OneHotEncoder(handle_unknown='ignore')),
                         ['ocean_proximity']),
                         ('kneighborsregressor',
                                         KNeighborsRegressor()))])]
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [83]: 1 results_dict["imp + scaling + ohe + KNN"] = mean_std_cross_val_scores(
2 knn_pipe, X_train, y_train, return_train_score=True
3)

In [84]: 1 pd.DataFrame(results_dict).T

	fit_time	score_time	test_score	train_score
dummy	0.002 (+/- 0.001)	0.000 (+/- 0.000)	-0.001 (+/- 0.001)	0.000 (+/- 0.000)
imp + scaling + ohe + KNN	0.029 (+/- 0.005)	0.087 (+/- 0.074)	0.721 (+/- 0.012)	0.816 (+/- 0.006)

```
In [85]: 1 svr_pipe = make_pipeline(preprocessor, SVR())
2 results_dict["imp + scaling + ohe + SVR (default)"] = mean_std_cross_val
3     svr_pipe, X_train, y_train, return_train_score=True
4 )
```

```
In [86]: 1 pd.DataFrame(results_dict).T
```

		fit_time	score_time	test_score	train_score
	dummy	0.002 (+/- 0.001)	0.000 (+/- 0.000)	-0.001 (+/- 0.001)	0.000 (+/- 0.000)
	imp + scaling + ohe + KNN	0.029 (+/- 0.005)	0.087 (+/- 0.074)	0.721 (+/- 0.012)	0.816 (+/- 0.006)
	imp + scaling + ohe + SVR (default)	10.170 (+/- 0.184)	4.659 (+/- 0.107)	-0.049 (+/- 0.012)	-0.049 (+/- 0.001)

The results with `scikit-learn`'s default SVR hyperparameters are pretty bad (negative R^2 , worse than predicting the mean!).

What should we try for hyper-parameters?

```
In [87]: 1 svr_C_pipe = make_pipeline(preprocessor, SVR(C=10000))
2 results_dict["imp + scaling + ohe + SVR (C=10000)"] = mean_std_cross_val
3     svr_C_pipe, X_train, y_train, return_train_score=True
4 )
```

```
In [88]: 1 pd.DataFrame(results_dict).T
```

		fit_time	score_time	test_score	train_score
	dummy	0.002 (+/- 0.001)	0.000 (+/- 0.000)	-0.001 (+/- 0.001)	0.000 (+/- 0.000)
	imp + scaling + ohe + KNN	0.029 (+/- 0.005)	0.087 (+/- 0.074)	0.721 (+/- 0.012)	0.816 (+/- 0.006)
	imp + scaling + ohe + SVR (default)	10.170 (+/- 0.184)	4.659 (+/- 0.107)	-0.049 (+/- 0.012)	-0.049 (+/- 0.001)
	imp + scaling + ohe + SVR (C=10000)	9.368 (+/- 0.159)	4.685 (+/- 0.171)	0.721 (+/- 0.007)	0.726 (+/- 0.007)

With a bigger value for `C` the results are much better. We need to carry out systematic hyperparameter optimization to get better results. (Coming up next week.)

- Note that categorical features are different than free text features. Sometimes there are columns containing free text information and we'll look at ways to deal with them in the later part of this lecture.

OHE with many categories

- Do we have enough data for rare categories to learn anything meaningful?
- How about grouping them into bigger categories?
 - Example: country names into continents such as "South America" or "Asia"
- Or having "other" category for rare cases?

Do we actually want to use certain features for prediction?

- Do you want to use certain features such as **gender** or **race** in prediction?
- Remember that the systems you build are going to be used in some applications.
- It's extremely important to be mindful of the consequences of including certain features in your predictive model.

Preprocessing the targets?

- Generally no need for this when doing classification.
- In regression it makes sense in some cases. More on this later.
- `sklearn` is fine with categorical labels (y -values) for classification problems.

?? Questions for you

True/False: Categorical features

1. `handle_unknown="ignore"` would treat all unknown categories equally.
2. Creating groups of rarely occurring categories might overfit the model.

Encoding text data

```
In [90]: 1 toy_spam = [
2     [
3         "URGENT!! As a valued network customer you have been selected to receive a £900 prize reward!",
4         "spam",
5     ],
6     [
7         "Lol you are always so convincing.", "non spam",
8         "Nah I don't think he goes to usf, he lives around here though",
9     ],
10    [
11        "URGENT! You have won a 1 week FREE membership in our £100000 prize Jackpot!",
12        "spam",
13    ],
14    [
15        "Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030",
16        "spam",
17    ],
18    [
19        "Congrats! I can't wait to see you!!", "non spam"
20    ]
21 toy_df = pd.DataFrame(toy_spam, columns=["sms", "target"])
```

Spam/non spam toy example

- What if the feature is in the form of raw text?
- The feature `sms` below is neither categorical nor ordinal.
- How can we encode it so that we can pass it to the machine learning algorithms we have seen so far?

```
In [91]: 1 toy_df
```

Out[91]:

		sms	target
0	URGENT!! As a valued network customer you have been selected to receive a £900 prize reward!		spam
1	Lol you are always so convincing.		non spam
2	Nah I don't think he goes to usf, he lives around here though		non spam
3	URGENT! You have won a 1 week FREE membership in our £100000 prize Jackpot!		spam
4	Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030		spam
5	Congrats! I can't wait to see you!!		non spam

What if we apply OHE?

In [92]:

```

1 ### DO NOT DO THIS.
2 enc = OneHotEncoder(sparse=False)
3 transformed = enc.fit_transform(toy_df[["sms"]])
4 pd.DataFrame(transformed, columns=enc.categories_)

```

Out[92]:

	Congrats! I can't wait to see you!!	Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030	Lol you are always so convincing.	Nah I don't think he goes to usf, he lives around here though	URGENT! You have won a 1 week FREE membership in our £100000 prize Jackpot!	URGENT!! As a valued network customer you have been selected to receive a £900 prize reward!
0	0.0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	1.0	0.0	0.0	0.0
2	0.0	0.0	0.0	1.0	0.0	0.0
3	0.0	0.0	0.0	0.0	1.0	0.0
4	0.0	1.0	0.0	0.0	0.0	0.0
5	1.0	0.0	0.0	0.0	0.0	0.0

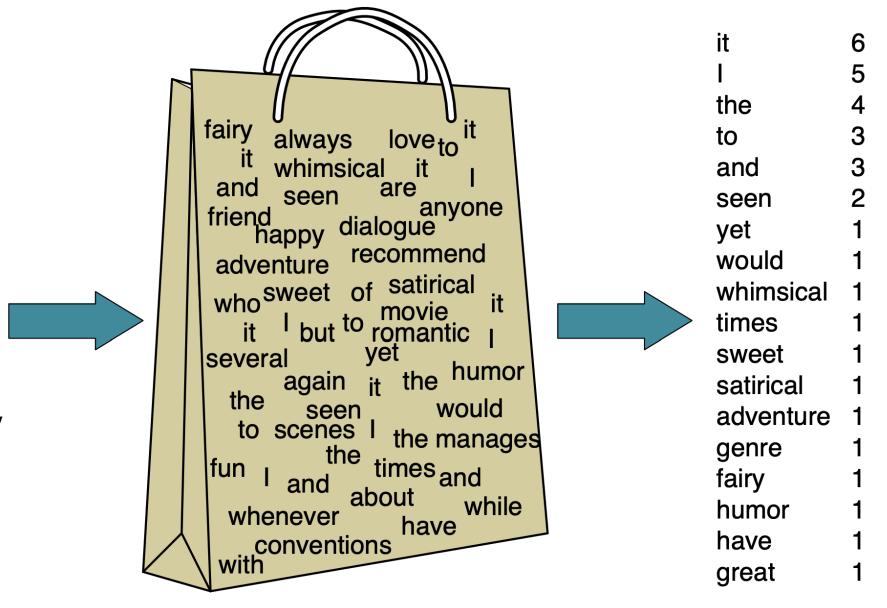
- We do not have a fixed number of categories here.
- Each "category" (feature value) is likely to occur only once in the training data and we won't learn anything meaningful if we apply one-hot encoding or ordinal encoding on this feature.

- How can we encode or represent raw text data into fixed number of features so that we can learn some useful patterns from it?
- This is a well studied problem in the field of Natural Language Processing (NLP), which is concerned with giving computers the ability to understand written and spoken language.
- Some popular representations of raw text include:
 - **Bag of words**
 - TF-IDF
 - Embedding representations

Bag of words (BOW) representation

- One of the most popular representation of raw text
- Ignores the syntax and word order
- It has two components:
 - The vocabulary (all unique words in all documents)
 - A value indicating either the presence or absence or the count of each word in the document.

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



Extracting BOW features using scikit-learn

- CountVectorizer
 - Converts a collection of text documents to a matrix of word counts.
 - Each row represents a "document" (e.g., a text message in our example).
 - Each column represents a word in the vocabulary (the set of unique words) in the training data.
 - Each cell represents how often the word occurs in the document.

In the Natural Language Processing (NLP) community text data is referred to as a **corpus** (plural: corpora).

In [93]:

```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 vec = CountVectorizer()
4 X_counts = vec.fit_transform(toy_df[ "sms" ])
5 bow_df = pd.DataFrame(
6     X_counts.toarray(), columns=vec.get_feature_names_out(), index=toy_
7 )
8 bow_df
```

Out[93]:

08002986030 100000 11 900 always are around as been call ... update urgen

sms	0	0	0	1	0	0	0	1	1	0	...	0	
URGENT!! As a valued network customer you have been selected to receive a £900 prize reward!	0	0	0	1	0	0	0	1	1	0	...	0	
Lol you are always so convincing.	0	0	0	0	1	1	0	0	0	0	...	0	1
Nah I don't think he goes to usf, he lives around here though	0	0	0	0	0	0	1	0	0	0	...	0	1
URGENT! You have won a 1 week FREE membership in our £100000 prize Jackpot!	0	1	0	0	0	0	0	0	0	0	...	0	
Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030	1	0	1	0	0	0	0	0	0	1	...	2	1
Congrats! I can't wait to see you!!	0	0	0	0	0	0	0	0	0	0	...	0	1

6 rows × 61 columns

In [94]: 1 type(toy_df["sms"])

Out[94]: pandas.core.series.Series

Note that unlike other transformers we are passing a `Series` object to `fit_transform`. For other transformers, you can define one transformer for more than one columns. But with `CountVectorizer` you need to define separate `CountVectorizer` transformers for each text column, if you have more than one text columns.

In [95]: 1 X_counts

Out[95]: <6x61 sparse matrix of type '<class 'numpy.int64'>'
with 71 stored elements in Compressed Sparse Row format>

Why sparse matrices?

- Most words do not appear in a given document.
- We get massive computational savings if we only store the nonzero elements.
- There is a bit of overhead, because we also need to store the locations:
 - e.g. "location (3,27): 1".
- However, if the fraction of nonzero is small, this is a huge win.

In [96]:

```

1 print("The total number of elements: ", np.prod(X_counts.shape))
2 print("The number of non-zero elements: ", X_counts.nnz)
3 print(
4     "Proportion of non-zero elements: %0.4f" % (X_counts.nnz / np.prod(
5 ))
6 print(
7     "The value at cell 3,%d is: %d"
8     % (vec.vocabulary_["jackpot"], X_counts[3, vec.vocabulary_["jackpot"]]
9 )

```

The total number of elements: 366
 The number of non-zero elements: 71
 Proportion of non-zero elements: 0.1940
 The value at cell 3,27 is: 1

Question for you

- What would happen if you apply StandardScaler on sparse data?

OneHotEncoder and sparse features

- By default, OneHotEncoder also creates sparse features.
- You could set `sparse=False` to get a regular numpy array.
- If there are a huge number of categories, it may be beneficial to keep them sparse.
- For smaller number of categories, it doesn't matter much.

Important hyperparameters of CountVectorizer

- `binary`
 - whether to use absence/presence feature values or counts
- `max_features`
 - only consider top `max_features` ordered by frequency in the corpus
- `max_df`
 - ignore features which occur in more than `max_df` documents
- `min_df`
 - ignore features which occur in less than `min_df` documents
- `ngram_range`
 - consider word sequences in the given range

Let's look at all features, i.e., words (along with their frequencies).

In [97]:

```
1 vec = CountVectorizer()
2 X_counts = vec.fit_transform(toy_df[ "sms" ])
3 bow_df = pd.DataFrame(
4     X_counts.toarray(), columns=vec.get_feature_names_out(), index=toy_
5 )
6 bow_df
```

Out[97]:

08002986030 100000 11 900 always are around as been call ... update urgen

sms	0	0	0	1	0	0	0	1	1	0	...	0	
URGENT!! As a valued network customer you have been selected to receive a £900 prize reward!	0	0	0	1	0	0	0	1	1	0	...	0	
Lol you are always so convincing.	0	0	0	0	1	1	0	0	0	0	...	0	1
Nah I don't think he goes to usf, he lives around here though	0	0	0	0	0	0	1	0	0	0	...	0	1
URGENT! You have won a 1 week FREE membership in our £100000 prize Jackpot!	0	1	0	0	0	0	0	0	0	0	...	0	
Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030	1	0	1	0	0	0	0	0	0	1	...	2	1
Congrats! I can't wait to see you!!	0	0	0	0	0	0	0	0	0	0	...	0	1

6 rows × 61 columns

When we use `binary=True`, the representation uses presence/absence of words instead of word counts.

In [98]:

```
1 vec_binary = CountVectorizer(binary=True)
2 X_counts = vec_binary.fit_transform(toy_df[ "sms" ])
3 bow_df = pd.DataFrame(
4     X_counts.toarray(), columns=vec_binary.get_feature_names_out(), ind
5 )
6 bow_df
```

Out[98]:

08002986030 100000 11 900 always are around as been call ... update urgen

sms	0	0	0	1	0	0	0	1	1	0	...	0	
URGENT!! As a valued network customer you have been selected to receive a £900 prize reward!	0	0	0	1	0	0	0	1	1	0	...	0	
Lol you are always so convincing.	0	0	0	0	1	1	0	0	0	0	...	0	1
Nah I don't think he goes to usf, he lives around here though	0	0	0	0	0	0	1	0	0	0	...	0	1
URGENT! You have won a 1 week FREE membership in our £100000 prize Jackpot!	0	1	0	0	0	0	0	0	0	0	...	0	
Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030	1	0	1	0	0	0	0	0	0	1	...	1	1
Congrats! I can't wait to see you!!	0	0	0	0	0	0	0	0	0	0	...	0	1

6 rows × 61 columns

We can control the size of X (the number of features) using `max_features`.

In [99]:

```

1 vec8 = CountVectorizer(max_features=8)
2 X_counts = vec8.fit_transform(toy_df[ "sms" ])
3 bow_df = pd.DataFrame(
4     X_counts.toarray(), columns=vec8.get_feature_names_out(), index=toy
5 )
6 bow_df

```

Out[99]:

	free	have	mobile	the	to	update	urgent	you
sms								
URGENT!! As a valued network customer you have been selected to receive a £900 prize reward!	0	1	0	0	1	0	1	1
Lol you are always so convincing.	0	0	0	0	0	0	0	1
Nah I don't think he goes to usf, he lives around here though	0	0	0	0	1	0	0	0
URGENT! You have won a 1 week FREE membership in our £100000 prize Jackpot!	1	1	0	0	0	0	1	1
Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030	2	0	2	2	2	2	0	0
Congrats! I can't wait to see you!!	0	0	0	0	1	0	0	1

Notice that `vec8` and `vec8_binary` have different vocabularies, which is kind of unexpected behaviour and doesn't match the documentation of `scikit-learn`.

[Here \(\[https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/feature_extraction/text.py#L1206-L1225\]\(https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/feature_extraction/text.py#L1206-L1225\)\)](https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/feature_extraction/text.py#L1206-L1225) is the code for `binary=True` condition in `scikit-learn`. As we can see, the binarization is done before limiting the features to `max_features`, and so now we are actually looking at the document counts (in how many documents it occurs) rather than term count. This is not explained anywhere in the documentation.

The ties in counts between different words makes it even more confusing. I don't think it'll have a big impact on the results but this is good to know! Remember that `scikit-learn` developers are also humans who are prone to make mistakes. So it's always a good habit to question whatever tools we use every now and then.

```
In [ ]: 1 vec8 = CountVectorizer(max_features=8)
2 X_counts = vec8.fit_transform(toy_df[ "sms" ])
3 pd.DataFrame(
4     data=X_counts.sum(axis=0).tolist()[0],
5     index=vec8.get_feature_names_out(),
6     columns=[ "counts" ],
7 ).sort_values( "counts" , ascending=False)
```

```
In [ ]: 1 vec8_binary = CountVectorizer(binary=True, max_features=8)
2 X_counts = vec8_binary.fit_transform(toy_df[ "sms" ])
3 pd.DataFrame(
4     data=X_counts.sum(axis=0).tolist()[0],
5     index=vec8_binary.get_feature_names_out(),
6     columns=[ "counts" ],
7 ).sort_values( "counts" , ascending=False)
```

Preprocessing

- Note that `CountVectorizer` is carrying out some preprocessing when used with default argument values, such as:
 - Converting words to lowercase (`lowercase=True`)
 - getting rid of punctuation and special characters (`token_pattern = '(?u)\\b\\w+\\b'`)

```
In [100]: 1 pipe = make_pipeline(CountVectorizer(), SVC())
```

```
In [101]: 1 pipe.fit(toy_df[ "sms" ], toy_df[ "target" ])
```

Out[101]: Pipeline(steps=[('countvectorizer', CountVectorizer()), ('svc', SVC())])
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [102]: 1 pipe.predict(toy_df[ "sms" ])
```

Out[102]: array(['spam', 'non spam', 'non spam', 'spam', 'spam', 'non spam'], dtype=object)

Is this a realistic representation of text data?

- Of course this is not a great representation of language
 - We are throwing out everything we know about language and losing a lot of information.
 - It assumes that there is no syntax and compositional meaning in language.
- But it works surprisingly well for many tasks.
- We will learn more expressive representations in the coming weeks.

? ? Questions for you

CountVectorizer: True or False

1. As you increase the value for `max_features` hyperparameter of `CountVectorizer` the training score is likely to go up.
2. Suppose you are encoding text data using `CountVectorizer`. If you encounter a word in the validation or the test split that's not available in the training data, we'll get an error.
3. `max_df` hyperparameter of `CountVectorizer` can be used to get rid of most frequently occurring words from the dictionary.
4. In the code below, inside `cross_validate`, each fold might have slightly different number of features (columns) in the fold.

```
pipe = (CountVectorizer(), SVC())
cross_validate(pipe, X_train, y_train)
```

Identify column transformations

Consider the restaurant data from the survey you did a few weeks ago.

```
In [ ]: 1 restaurant_data = pd.read_csv("../data/cleaned_restaurant_data.csv")
2 restaurant_data.head(10)
```

What all feature transformations you would apply on this dataset?

What did we learn today?

- Motivation to use `ColumnTransformer`
- `ColumnTransformer` syntax
- Defining transformers with multiple transformations
- How to visualize transformed features in a dataframe
- More on ordinal features
- Different arguments `OneHotEncoder`
 - `handle_unknown="ignore"`
 - `if_binary`
- Dealing with text features

- Bag of words representation: CountVectorizer



CPSC 330

Applied Machine Learning

Lecture 7: Linear Models

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

```
In [2]: 1 import os
2 import sys
3
4 sys.path.append("../code/.")
5
6 import IPython
7 import ipywidgets as widgets
8 import matplotlib.pyplot as plt
9 import mglearn
10 import numpy as np
11 import pandas as pd
12 from IPython.display import HTML, display
13 from ipywidgets import interact, interactive
14 from plotting_functions import *
15 from sklearn.dummy import DummyClassifier
16 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
17 from sklearn.impute import SimpleImputer
18 from sklearn.model_selection import cross_val_score, cross_validate, train_test_split
19 from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
20 from sklearn.pipeline import Pipeline, make_pipeline
21 from sklearn.preprocessing import OneHotEncoder, StandardScaler
22 from sklearn.impute import SimpleImputer
23 from sklearn.svm import SVC
24 from sklearn.tree import DecisionTreeClassifier
25 from sklearn.compose import make_column_transformer
26 from utils import *
27
28 %matplotlib inline
29 pd.set_option("display.max_colwidth", 200)
```

Learning outcomes

From this lecture, students are expected to be able to:

- Explain how `predict` works for linear regression;
- Use `scikit-learn`'s Ridge model;
- Demonstrate how the `alpha` hyperparameter of Ridge is related to the fundamental tradeoff;
- Explain the difference between linear regression and logistic regression;
- Use `scikit-learn`'s `LogisticRegression` model and `predict_proba` to get probability scores
- Explain the advantages of getting probability scores instead of hard predictions during classification;
- Broadly describe linear SVMs
- Explain how one can interpret model predictions using coefficients learned by a linear model;
- Explain the advantages and limitations of linear classifiers
- Carry out multi-class classification using One-Vs-Rest(All) and One-Vs-One strategies (later, Lecture 17).

Linear models [[video \(<https://youtu.be/HXd1U2q4VFA>\)](https://youtu.be/HXd1U2q4VFA)]

Linear models is a fundamental and widely used class of models. They are called **linear** because they make a prediction using a **linear function** of the input features.

We will talk about three linear models:

- Linear regression
- Logistic regression
- Linear SVM (brief mention)

Linear regression

- A very popular statistical model and has a long history.
- Imagine a hypothetical regression problem of predicting the weight of a snake given its length.

In [3]:

```
1 np.random.seed(7)
2 n = 100
3 X_1 = np.linspace(0, 2, n) + np.random.randn(n) * 0.01
4 X = pd.DataFrame(X_1[:, None], columns=["length"])
5
6 y = abs(np.random.randn(n, 1)) * 3 + X_1[:, None] * 5 + 0.2
7 y = pd.DataFrame(y, columns=["weight"])
8 snakes_df = pd.concat([X, y], axis=1)
9 train_df, test_df = train_test_split(snakes_df, test_size=0.2, random_s
10
11 X_train = train_df[["length"]]
12 y_train = train_df["weight"]
13 X_test = test_df[["length"]]
14 y_test = test_df["weight"]
15 train_df.head()
```

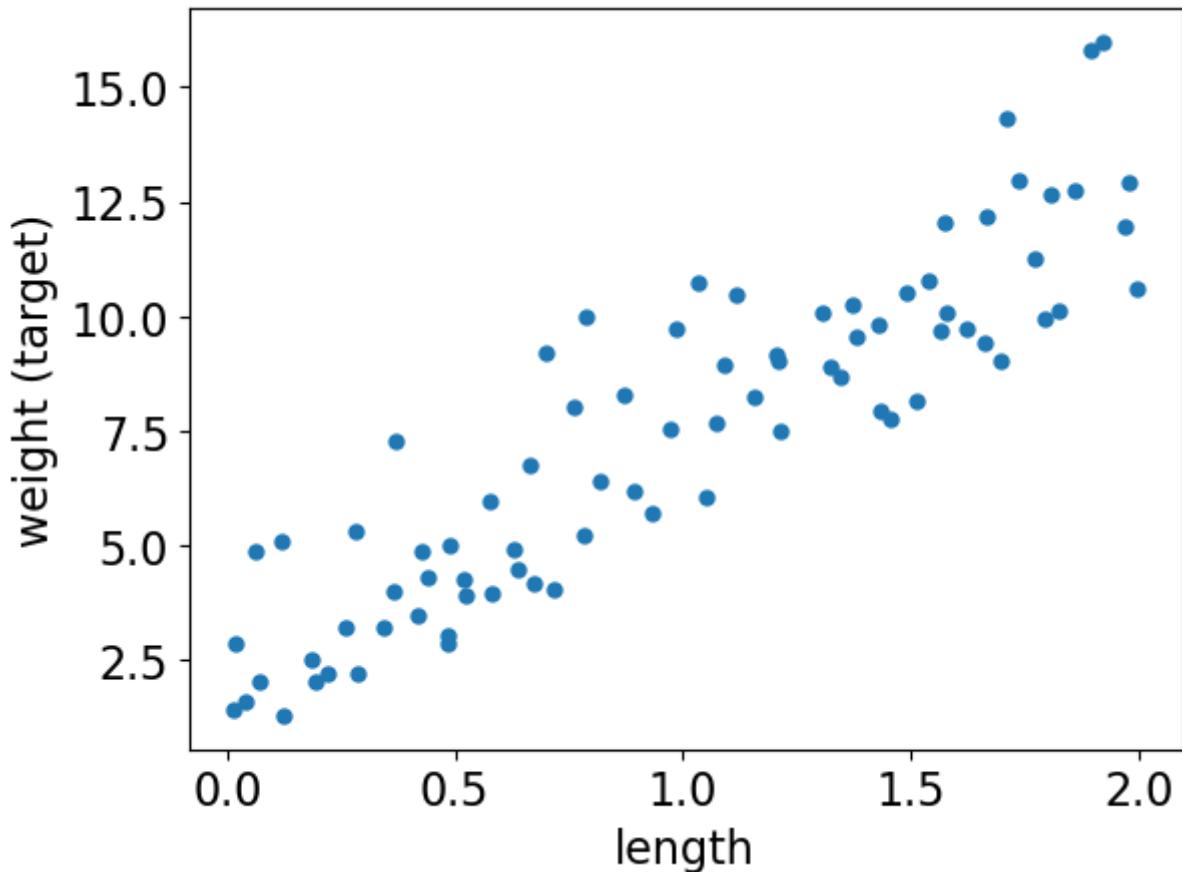
Out[3]:

	length	weight
73	1.489130	10.507995
53	1.073233	7.658047
80	1.622709	9.748797
49	0.984653	9.731572
23	0.484937	3.016555

Let's visualize the hypothetical snake data.

In [4]:

```
1 plt.plot(X_train, y_train, ".", markersize=10)
2 plt.xlabel("length")
3 plt.ylabel("weight (target)");
```



Let's plot a linear regression model on this dataset.

In [5]:

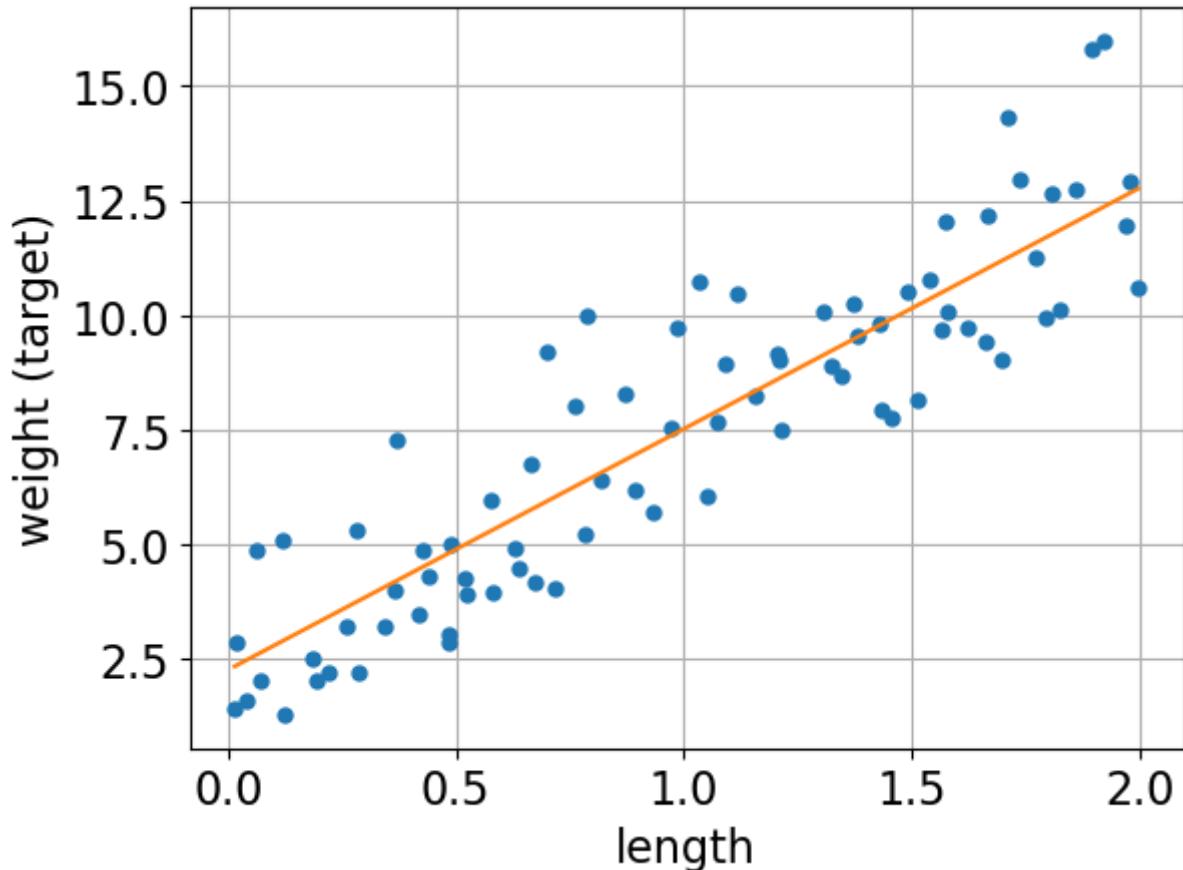
```
1 grid = np.linspace(min(X_train.to_numpy())[0], max(X_train.to_numpy())[0])
2 grid = grid.reshape(-1, 1)
```

In [6]:

```

1 from sklearn.linear_model import Ridge
2
3 r = Ridge()
4 r.fit(X_train.to_numpy(), y_train)
5 plt.plot(X_train, y_train, ".", markersize=10)
6 plt.plot(grid, r.predict(grid))
7 plt.grid(True)
8 plt.xlabel("length")
9 plt.ylabel("weight (target)");

```



The orange line is the learned linear model.

Prediction of linear regression

- Given a snake length, we can use the model above to predict the target (i.e., the weight of the snake).
- The prediction will be the corresponding weight on the orange line.

In [7]:

```

1 snake_length = 0.75
2 r.predict([[snake_length]])

```

Out[7]: array([6.20683258])

What are we exactly learning?

- The model above is a line, which can be represented with a slope (i.e., coefficient or weight) and an intercept.
- For the above model, we can access the slope (i.e., coefficient or weight) and the intercept using `coef_` and `intercept_`, respectively.

```
In [8]: 1 r.coef_ # r is our linear regression object
```

Out[8]: array([5.26370005])

```
In [9]: 1 r.intercept_ # r is our linear regression object
```

Out[9]: 2.259057547817185

How are we making predictions?

- Given a feature value x_1 and learned coefficient w_1 and intercept b , we can get the prediction \hat{y} with the following formula:

$$\hat{y} = w_1 x_1 + b$$

```
In [10]: 1 prediction = snake_length * r.coef_ + r.intercept_
2 prediction
```

Out[10]: array([6.20683258])

```
In [11]: 1 r.predict([[snake_length]])
```

Out[11]: array([6.20683258])

Great! Now we exactly know how the model is making the prediction.

Generalizing to more features

For more features, the model is a higher dimensional hyperplane and the general prediction formula looks as follows:

$$\hat{y} = w_1 x_1 + \dots + w_d x_d + b$$

where,

- (x_1, \dots, x_d) are input features
- (w_1, \dots, w_d) are coefficients or weights (learned from the data)
- b is the bias which can be used to offset your hyperplane (learned from the data)

Example

- Suppose these are the coefficients learned by a linear regression model on a hypothetical housing price prediction dataset.

Feature	Learned coefficient
Bedrooms	0.20
Bathrooms	0.11
Square Footage	0.002
Age	-0.02

- Now given a new example, the target will be predicted as follows:

Bedrooms	Bathrooms	Square Footage	Age
3	2	1875	66

$$\hat{y} = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

$$\text{predicted price} = 0.20 \times 3 + 0.11 \times 2 + 0.002 \times 1875 + (-0.02) \times 66 + b$$

When we call `fit`, a coefficient or weight is learned for each feature which tells us the role of that feature in prediction. These coefficients are learned from the training data.

In linear models for regression, the model is a line for a single feature, a plane for two features, and a hyperplane for higher dimensions. We are not yet ready to discuss how does linear regression learn these coefficients and intercept.

Ridge

- `scikit-learn` has a model called `LinearRegression` for linear regression.
- But if we use this "vanilla" version of linear regression, it may result in large coefficients and unexpected results.
- So instead of using `LinearRegression`, we will always use another linear model called `Ridge`, which is a linear regression model with a complexity hyperparameter `alpha`.
- If you want to know more about how Ridge regularization works, we recommend this tutorial: <https://www.analyticsvidhya.com/blog/2016/01/ridge-lasso-regression-python-complete-tutorial/> (<https://www.analyticsvidhya.com/blog/2016/01/ridge-lasso-regression-python-complete-tutorial/>)

In [12]:

```
1 from sklearn.linear_model import LinearRegression # DO NOT USE IT
2 from sklearn.linear_model import Ridge # USE THIS INSTEAD
```

Data


```
In [13]: 1 housing_df = pd.read_csv("../data/housing.csv")
2 train_df, test_df = train_test_split(housing_df, test_size=0.1, random_
3
4 train_df = train_df.assign(
5     rooms_per_household=train_df["total_rooms"] / train_df["households"]
6 )
7 test_df = test_df.assign(
8     rooms_per_household=test_df["total_rooms"] / test_df["households"]
9 )
10
11 train_df = train_df.assign(
12     bedrooms_per_household=train_df["total_bedrooms"] / train_df["house-
13 ")
14 test_df = test_df.assign(
15     bedrooms_per_household=test_df["total_bedrooms"] / test_df["househo-
16 ")
17
18 train_df = train_df.assign(
19     population_per_household=train_df["population"] / train_df["househo-
20 ")
21 test_df = test_df.assign(
22     population_per_household=test_df["population"] / test_df["household-
23 ")
24
25 X_train = train_df.drop(columns=["median_house_value"])
26 y_train = train_df["median_house_value"]
27
28 X_test = test_df.drop(columns=["median_house_value"])
29 y_test = test_df["median_house_value"]
30
31 numeric_features = [
32     "housing_median_age",
33     "population",
34     "households",
35     "median_income",
36     "rooms_per_household",
37     "bedrooms_per_household",
38     "population_per_household",
39 ]
40 categorical_features = ["ocean_proximity"]
41 drop_features = [
42     "longitude",
43     "latitude",
44     "total_rooms",
45     "total_bedrooms",
46 ]
47
48 ct = make_column_transformer(
49     (
50         make_pipeline(SimpleImputer(strategy="median"), StandardScaler(
51             numeric_features,
52         )),
53         (
54             OneHotEncoder(handle_unknown="ignore"),
55             categorical_features,
56         ),
57         ("drop", drop_features),

```

58)

In [14]:

```
1 pipe = make_pipeline(ct, Ridge())
2 scores = cross_validate(pipe, X_train, y_train, return_train_score=True)
3 pd.DataFrame(scores)
```

Out[14]:

	fit_time	score_time	test_score	train_score
0	0.038253	0.007985	0.622840	0.634925
1	0.031309	0.006604	0.651388	0.627447
2	0.034430	0.006995	0.635000	0.632330
3	0.033828	0.007207	0.622482	0.635518
4	0.035256	0.008641	0.595064	0.635765

Hyperparameter alpha of Ridge

- Ridge has hyperparameters just like the rest of the models we learned.
- The alpha hyperparameter is what makes Ridge different from vanilla LinearRegression .
- Similar to the other hyperparameters that we saw, alpha controls the fundamental tradeoff.

If we set alpha=0 that is the same as using LinearRegression .

Let's examine the effect of alpha on the fundamental tradeoff.

In [15]:

```
1 scores_dict = {
2     "alpha": 10.0 ** np.arange(-2, 6, 1),
3     "mean_train_scores": list(),
4     "mean_cv_scores": list(),
5 }
6 for alpha in scores_dict["alpha"]:
7     pipe_ridge = make_pipeline(ct, Ridge(alpha=alpha))
8     scores = cross_validate(pipe_ridge, X_train, y_train, return_train_
9     scores_dict["mean_train_scores"].append(scores["train_score"].mean()
10    scores_dict["mean_cv_scores"].append(scores["test_score"].mean())
11
12 results_df = pd.DataFrame(scores_dict)
```

In [16]: 1 results_df

Out[16]:

	alpha	mean_train_scores	mean_cv_scores
0	0.01	0.633217	0.625408
1	0.10	0.633216	0.625405
2	1.00	0.633197	0.625355
3	10.00	0.632909	0.624734
4	100.00	0.632352	0.620946
5	1000.00	0.621155	0.578949
6	10000.00	0.490031	0.438445
7	100000.00	0.140880	0.137042

Here we do not really see overfitting but in general,

- larger alpha → likely to underfit
- smaller alpha → likely to overfit

Coefficients and intercept

The model learns

- coefficients associated with each feature
- the intercept or bias

Let's examine the coefficients learned by the model.

In [17]:

```
1 pipe_ridge = make_pipeline(ct, Ridge(alpha=10.0))
2 pipe_ridge.fit(X_train, y_train)
3 coeffs = pipe_ridge.named_steps["ridge"].coef_
```

In [18]:

```

1 column_names = numeric_features + list(
2     ct.named_transformers_["onehotencoder"].get_feature_names_out(
3         categorical_features
4     )
5 )
6
7 pd.DataFrame(data=coeffs, index=column_names, columns=["Coefficients"])

```

Out[18]:

	Coefficients
housing_median_age	14658.019489
population	-43890.006653
households	49138.464132
median_income	78819.238154
rooms_per_household	-15062.467094
bedrooms_per_household	17543.750856
population_per_household	625.875537
ocean_proximity_<1H OCEAN	-1660.944525
ocean_proximity_INLAND	-68031.703729
ocean_proximity_ISLAND	56468.596208
ocean_proximity_NEAR BAY	1529.230447
ocean_proximity_NEAR OCEAN	11694.821599

- The model also learns an intercept (bias).
- For each prediction, we are adding this amount irrespective of the feature values.

In [19]:

```
1 pipe_ridge.named_steps["ridge"].intercept_
```

Out[19]:

227002.23569246812

Can we use this information to interpret model predictions?

?? Questions for you

True/False: Ridge

1. Increasing the hyperparameter alpha of Ridge is likely to decrease model complexity.
2. Ridge can be used with datasets that have multiple features.
3. With Ridge, we learn one coefficient per training example.
4. If you train a linear regression model on a 2-dimensional problem (2 features), the model will be a two dimensional plane.

Interpretation of coefficients

- One of the main advantages of linear models is that they are relatively easy to interpret.
- We have one coefficient per feature which kind of describes the role of the feature in the prediction according to the model.

There are two pieces of information in the coefficients based on

- Sign
- Magnitude

Sign of the coefficients

```
In [20]: 1 pd.DataFrame(data=coeffs, index=column_names, columns=["Coefficients"])
```

Out[20]:

	Coefficients
housing_median_age	14658.019489
population	-43890.006653
households	49138.464132
median_income	78819.238154
rooms_per_household	-15062.467094
bedrooms_per_household	17543.750856
population_per_household	625.875537
ocean_proximity_<1H OCEAN	-1660.944525
ocean_proximity_INLAND	-68031.703729
ocean_proximity_ISLAND	56468.596208
ocean_proximity_NEAR BAY	1529.230447
ocean_proximity_NEAR OCEAN	11694.821599

Magnitude of the coefficients

- Bigger magnitude → bigger impact on the prediction
- In the example below, both RM and AGE have a positive impact on the prediction but RM would have a bigger positive impact because its feature value is going to be multiplied by a number with a bigger magnitude.

- Similarly both LSAT and NOX have a negative impact on the prediction but LSAT would have a bigger negative impact because it's going to be multiplied by a number with a bigger magnitude.

In [21]:

```

1 data = {
2     "coefficient": pipe_ridge.named_steps["ridge"].coef_.tolist(),
3     "magnitude": np.absolute(pipe_ridge.named_steps["ridge"].coef_.tolist())
4 }
5 coef_df = pd.DataFrame(data, index=column_names).sort_values(
6     "magnitude", ascending=False
7 )
8 coef_df

```

Out[21]:

	coefficient	magnitude
median_income	78819.238154	78819.238154
ocean_proximity_INLAND	-68031.703729	68031.703729
ocean_proximity_ISLAND	56468.596208	56468.596208
households	49138.464132	49138.464132
population	-43890.006653	43890.006653
bedrooms_per_household	17543.750856	17543.750856
rooms_per_household	-15062.467094	15062.467094
housing_median_age	14658.019489	14658.019489
ocean_proximity_NEAR OCEAN	11694.821599	11694.821599
ocean_proximity_<1H OCEAN	-1660.944525	1660.944525
ocean_proximity_NEAR BAY	1529.230447	1529.230447
population_per_household	625.875537	625.875537

Importance of scaling

- When you are interpreting the model coefficients, scaling is crucial.
- If you do not scale the data, features with smaller magnitude are going to get coefficients with bigger magnitude whereas features with bigger scale are going to get coefficients with smaller magnitude.
- That said, when you scale the data, feature values become hard to interpret for humans!

Take these coefficients with a grain of salt. They might not always match your intuitions.

? ? Questions for you

True/False

1. Suppose you have trained a linear model on an unscaled data. The coefficients of the linear model have the following interpretation: if coefficient j is large, that means a change in feature j has a large impact on the prediction.
2. Suppose the scaled feature value of `population` above is negative. The prediction will still be inversely proportional to `population`; as `population` gets **bigger**, the median house value gets **smaller**.

Questions for breakout room discussion

- Discuss the importance of scaling when interpreting linear regression coefficients.
- What might be the meaning of complex vs simpler model in case of linear regression?

Logistic regression [[video](#) (<https://youtu.be/56L5zt22qE>)]

Logistic regression intuition

- A linear model for **classification**.
- Similar to linear regression, it learns weights associated with each feature and the bias.
- It applies a **threshold** on the raw output to decide whether the class is positive or negative.
- In this lecture we will focus on the following aspects of logistic regression.
 - `predict` , `predict_proba`
 - how to use learned coefficients to interpret the model

Motivating example

- Consider the problem of predicting sentiment expressed in movie reviews.

Training data for the motivating example

Review 1: This movie was **excellent!** The performances were oscar-worthy! 

Review 2: What a **boring** movie! I almost fell asleep twice while watching it. 

Review 3: I enjoyed the movie. **Excellent!** 

- Targets: positive  and negative 
- Features: words (e.g., *excellent*, *flawless*, *boring*)

Learned coefficients associated with all features

- Suppose our vocabulary contains only the following 7 words.
- A linear classifier learns **weights** or **coefficients** associated with the features (words in this example).
- Let's ignore bias for a bit.

Word	Coefficient
excellent	1.93
disappointment	-2.20
flawless	1.43
boring	-1.40
unwatchable	-2.04
incoherent	-1.86
subtle	1.30

Predicting with learned weights

- Use these learned coefficients to make predictions. For example, consider the following review x_i .

It got a bit **boring** at times but the direction was **excellent** and the acting was **flawless**.

- Feature vector for x_i : [1, 0, 1, 1, 0, 0, 0]

Word	Coefficient
excellent	1.93
disappointment	-2.20
flawless	1.43
boring	-1.40
unwatchable	-2.04
incoherent	-1.86
subtle	1.30

- $score(x_i) = \text{coefficient}(boring) \times 1 + \text{coefficient}(excellent) \times 1 + \text{coefficient}(flawless) \times 1 = -1.40 + 1.93 + 1.43 = 1.96$
- $1.96 > 0$ so predict the review as positive .

```
In [22]: 1 x = ["boring=1", "excellent=1", "flawless=1"]
2 w = [-1.40, 1.93, 1.43]
3 display(plot_logistic_regression(x, w))
```

Weighted sum of the input features = 1.960 y_hat = pos
<graphviz.graphs.Digraph at 0x18a688520>

- So the prediction is based on the weighted sum of the input features.
- Some feature are pulling the prediction towards positive sentiment and some are pulling it towards negative sentiment.
- If the coefficient of *boring* had a bigger magnitude or *excellent* and *flawless* had smaller magnitudes, we would have predicted "neg".

```
In [23]: 1 def f(w_0):
2     x = ["boring=1", "excellent=1", "flawless=1"]
3     w = [-1.40, 1.93, 1.43]
4     w[0] = w_0
5     print(w)
6     display(plot_logistic_regression(x, w))
```

```
In [24]: 1 interactive(
2     f,
3     w_0=widgets.FloatSlider(min=-6, max=2, step=0.5, value=-1.40),
4 )
```

Out[24]: `interactive(children=(FloatSlider(value=-1.4, description='w_0', max=2.0, min=-6.0, step=0.5), Output()), _dom...`

In our case, for values for the coefficient of *boring* < -3.36 , the prediction would be negative.

So a linear classifier is a linear function of the input x , followed by a threshold.

$$\begin{aligned} z &= w_1 x_1 + \cdots + w_d x_d + b \\ &= w^T x + b \end{aligned}$$

$$\hat{y} = \begin{cases} 1, & \text{if } z \geq r \\ -1, & \text{if } z < r \end{cases}$$

Components of a linear classifier

1. input features (x_1, \dots, x_d)
2. coefficients (weights) (w_1, \dots, w_d)
3. bias (b or w_0) (can be used to offset your hyperplane)
4. threshold (r)

In our example before, we assumed $r = 0$ and $b = 0$.

Logistic regression on the cities data

```
In [25]: 1 cities_df = pd.read_csv("../data/canada_usa_cities.csv")
2 train_df, test_df = train_test_split(cities_df, test_size=0.2, random_s
3 X_train, y_train = train_df.drop(columns=["country"], axis=1), train_df
4 X_test, y_test = test_df.drop(columns=["country"], axis=1), test_df["co
5
6 train_df.head()
```

```
Out[25]:    longitude  latitude  country
160     -76.4813   44.2307  Canada
127     -81.2496   42.9837  Canada
169     -66.0580   45.2788  Canada
188     -73.2533   45.3057  Canada
187     -67.9245   47.1652  Canada
```

Let's first try `DummyClassifier` on the cities data.

In [26]:

```
1 dummy = DummyClassifier()
2 scores = cross_validate(dummy, X_train, y_train, return_train_score=True)
3 pd.DataFrame(scores)
```

Out[26]:

	fit_time	score_time	test_score	train_score
0	0.001376	0.002050	0.588235	0.601504
1	0.000801	0.000412	0.588235	0.601504
2	0.001103	0.000663	0.606061	0.597015
3	0.000771	0.000396	0.606061	0.597015
4	0.000691	0.000353	0.606061	0.597015

Now let's try LogisticRegression

In [27]:

```
1 from sklearn.linear_model import LogisticRegression
2
3 lr = LogisticRegression()
4 scores = cross_validate(lr, X_train, y_train, return_train_score=True)
5 pd.DataFrame(scores)
```

Out[27]:

	fit_time	score_time	test_score	train_score
0	0.014193	0.001574	0.852941	0.827068
1	0.007429	0.001624	0.823529	0.827068
2	0.006945	0.001767	0.696970	0.858209
3	0.008119	0.001420	0.787879	0.843284
4	0.006647	0.001773	0.939394	0.805970

Logistic regression seems to be doing better than dummy classifier. But note that there is a lot of variation in the scores.

Accessing learned parameters

- Recall that logistic regression learns the weights w and bias or intercept b .
- How to access these weights?
 - Similar to Ridge, we can access the weights and intercept using `coef_` and `intercept_` attribute of the `LogisticRegression` object, respectively.

In [28]:

```

1 lr = LogisticRegression()
2 lr.fit(X_train.to_numpy(), y_train)
3 print("Model weights: %s" % (lr.coef_)) # these are the learned weight
4 print("Model intercept: %s" % (lr.intercept_)) # this is the bias term
5 data = {"features": X_train.columns, "coefficients": lr.coef_[0]}
6 pd.DataFrame(data)

```

Model weights: [[-0.04108149 -0.33683126]]
 Model intercept: [10.8869838]

Out[28]:

	features	coefficients
0	longitude	-0.041081
1	latitude	-0.336831

- Both negative weights
- The weight of latitude is larger in magnitude.
- This makes sense because Canada as a country lies above the USA and so we expect latitude values to contribute more to a prediction than longitude.

Prediction with learned parameters

Let's predict target of a test example.

In [29]:

```

1 example = x_test.iloc[0, :]
2 example

```

Out[29]:

longitude -64.8001
 latitude 46.0980
 Name: 172, dtype: float64

Raw scores

- Calculate the raw score as: $y_{\hat{}} = \mathbf{w} \cdot \mathbf{x} + b$

In [30]:

```

1 (
2     np.dot(
3         example.to_numpy(),
4         lr.coef_.reshape(
5             2,
6             ),
7         )
8     + lr.intercept_
9 )

```

Out[30]:

array([-1.97817876])

- Apply the threshold to the raw score.
- Since the prediction is < 0, predict "negative".

- What is a "negative" class in our context?
- With logistic regression, the model randomly assigns one of the classes as a positive class and the other as negative.
 - Usually it would alphabetically order the target and pick the first one as negative and second one as the positive class.

- The `classes_` attribute tells us which class is considered negative and which one is considered positive. - In this case, Canada is the negative class and USA is a positive class.

```
In [31]: 1 lr.classes_
```

```
Out[31]: array(['Canada', 'USA'], dtype=object)
```

- So based on the negative score above (-1.978), we would predict Canada.
- Let's check the prediction given by the model.

```
In [32]: 1 lr.predict([example])
```

```
Out[32]: array(['Canada'], dtype=object)
```

Great! The predictions match! We exactly know how the model is making predictions.

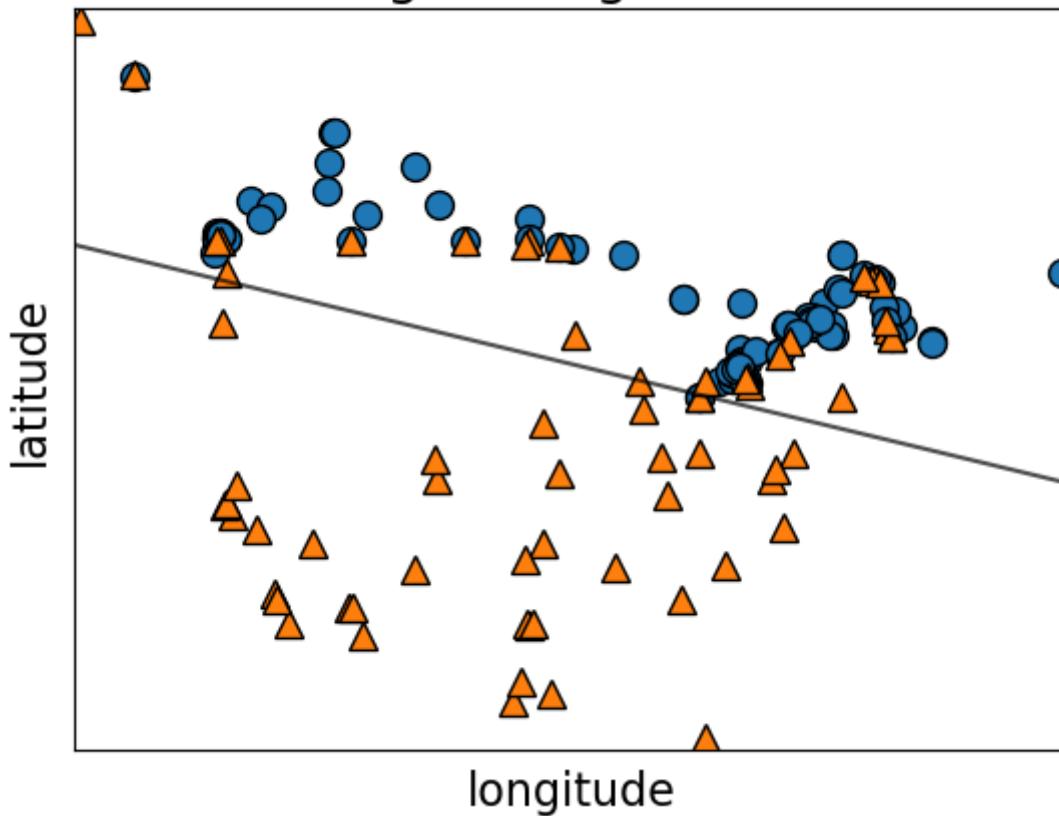
Decision boundary of logistic regression

- The decision boundary of logistic regression is a **hyperplane** dividing the feature space in half.

In [33]:

```
1 lr = LogisticRegression()
2 lr.fit(X_train.to_numpy(), y_train)
3 mglearn.discrete_scatter(X_train.iloc[:, 0], X_train.iloc[:, 1], y_train)
4 mglearn.plots.plot_2d_separator(lr, X_train.to_numpy(), fill=False, eps=0.5)
5 plt.title(lr.__class__.__name__)
6 plt.xlabel("longitude")
7 plt.ylabel("latitude");
```

LogisticRegression



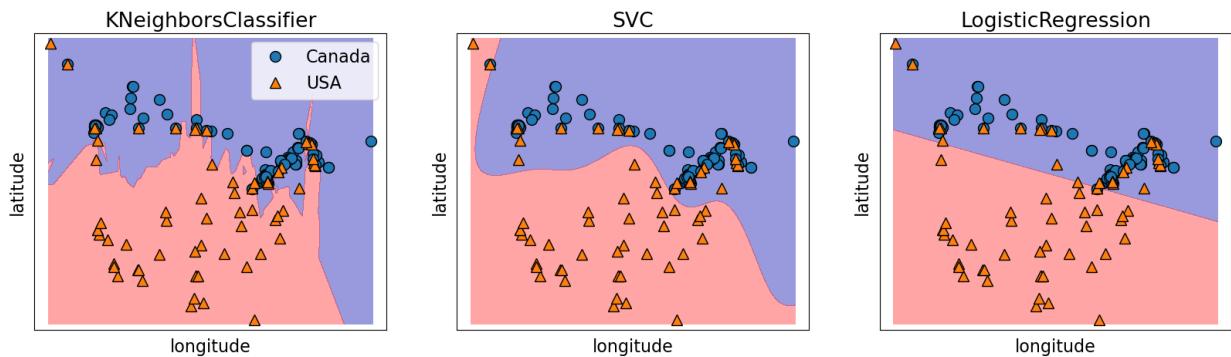
- For $d = 2$, the decision boundary is a line (1-dimensional)
- For $d = 3$, the decision boundary is a plane (2-dimensional)
- For $d > 3$, the decision boundary is a $d - 1$ -dimensional hyperplane

In [34]:

```

1 fig, axes = plt.subplots(1, 3, figsize=(20, 5))
2 for model, ax in zip(
3     [KNeighborsClassifier(), SVC(gamma=0.01), LogisticRegression()], axes):
4     clf = model.fit(X_train.to_numpy(), y_train)
5     mglearn.plots.plot_2d_separator(
6         clf, X_train.to_numpy(), fill=True, eps=0.5, ax=ax, alpha=0.4
7     )
8     mglearn.discrete_scatter(X_train.iloc[:, 0], X_train.iloc[:, 1], y_
9     ax.set_title(clf.__class__.__name__)
10    ax.set_xlabel("longitude")
11    ax.set_ylabel("latitude")
12 axes[0].legend();

```



- Notice a linear decision boundary (a line in our case).
- Compare it with KNN or SVM RBF decision boundaries.

Main hyperparameter of logistic regression

- C is the main hyperparameter which controls the fundamental trade-off.
- We won't really talk about the interpretation of this hyperparameter right now.
- At a high level, the interpretation is similar to C of SVM RBF
 - smaller C → might lead to underfitting
 - bigger C → might lead to overfitting

In [35]:

```

1 scores_dict = {
2     "C": 10.0 ** np.arange(-4, 6, 1),
3     "mean_train_scores": list(),
4     "mean_cv_scores": list(),
5 }
6 for C in scores_dict["C"]:
7     lr = LogisticRegression(C=C)
8     scores = cross_validate(lr, X_train, y_train, return_train_score=True)
9     scores_dict["mean_train_scores"].append(scores["train_score"].mean())
10    scores_dict["mean_cv_scores"].append(scores["test_score"].mean())
11
12 results_df = pd.DataFrame(scores_dict)
13 results_df

```

Out[35]:

	C	mean_train_scores	mean_cv_scores
0	0.0001	0.664707	0.658645
1	0.0010	0.784424	0.790731
2	0.0100	0.827842	0.826203
3	0.1000	0.832320	0.820143
4	1.0000	0.832320	0.820143
5	10.0000	0.832320	0.820143
6	100.0000	0.832320	0.820143
7	1000.0000	0.832320	0.820143
8	10000.0000	0.832320	0.820143
9	100000.0000	0.832320	0.820143

Predicting probability scores [video (https://youtu.be/_OAK5KiGLg0)]

`predict_proba`

- So far in the context of classification problems, we focused on getting "hard" predictions.
- Very often it's useful to know "soft" predictions, i.e., how confident the model is with a given prediction.
- For most of the `scikit-learn` classification models we can access this confidence score or probability score using a method called `predict_proba`.

Let's look at probability scores of logistic regression model for our test example.

In [36]: 1 example

Out[36]: longitude -64.8001
latitude 46.0980
Name: 172, dtype: float64

In [37]: 1 lr = LogisticRegression()
2 lr.fit(X_train.to_numpy(), y_train)
3 lr.predict([example]) # hard prediction

Out[37]: array(['Canada'], dtype=object)

In [38]: 1 lr.predict_proba([example]) # soft prediction

Out[38]: array([[0.87848688, 0.12151312]])

- The output of `predict_proba` is the probability of each class.
- In binary classification, we get probabilities associated with both classes (even though this information is redundant).
- The first entry is the estimated probability of the first class and the second entry is the estimated probability of the second class from `model.classes_`.

In [39]: 1 lr.classes_

Out[39]: array(['Canada', 'USA'], dtype=object)

- Because it's a probability, the sum of the entries for both classes should always sum to 1.
- Since the probabilities for the two classes sum to 1, exactly one of the classes will have a score ≥ 0.5 , which is going to be our predicted class.

How does logistic regression calculate these probabilities?

- The weighted sum $w_1x_1 + \dots + w_dx_d + b$ gives us "raw model output".
- For linear regression this would have been the prediction.
- For logistic regression, you check the **sign** of this value.
 - If positive (or 0), predict +1; if negative, predict -1.
 - These are "hard predictions".

- You can also have "soft predictions", aka **predicted probabilities**.
 - To convert the raw model output into probabilities, instead of taking the sign, we apply the **sigmoid**.

The sigmoid function

- The sigmoid function "squashes" the raw model output from any number to the range $[0, 1]$ using the following formula, where x is the raw model output.

$$\frac{1}{1 + e^{-x}}$$

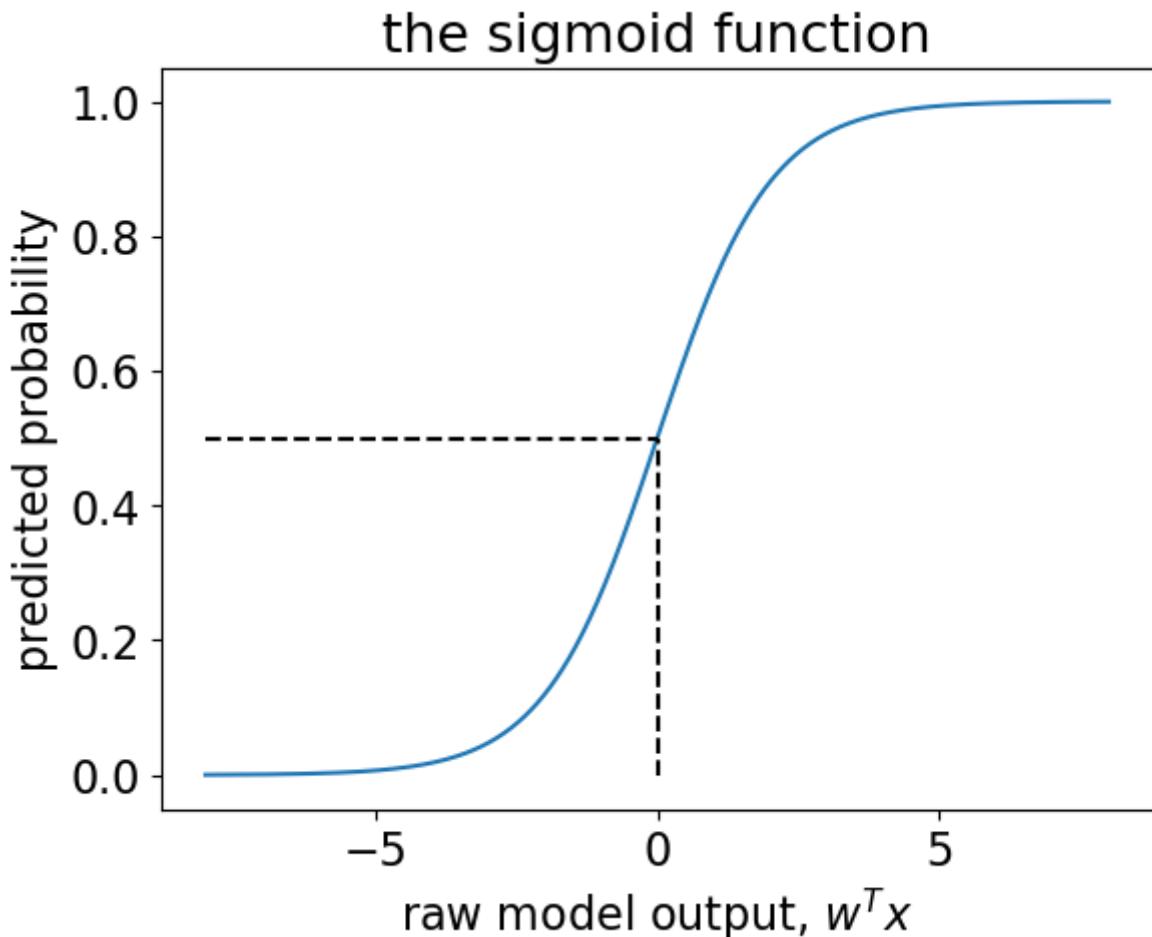
- Then we can interpret the output as probabilities.

In [40]:

```

1 sigmoid = lambda x: 1 / (1 + np.exp(-x))
2 raw_model_output = np.linspace(-8, 8, 1000)
3 plt.plot(raw_model_output, sigmoid(raw_model_output))
4 plt.plot([0, 0], [0, 0.5], "--k")
5 plt.plot([-8, 0], [0.5, 0.5], "--k")
6 plt.xlabel("raw model output, $w^T x$")
7 plt.ylabel("predicted probability")
8 plt.title("the sigmoid function");

```



- Recall our hard predictions that check the sign of $w^T x$, or, in other words, whether or not it is ≥ 0 .
 - The threshold $w^T x = 0$ corresponds to $p = 0.5$.
 - In other words, if our predicted probability is ≥ 0.5 then our hard prediction is $+1$.

Let's get the probability score by calling sigmoid on the raw model output for our test example.

In [41]:

```

1 sigmoid(
2     np.dot(
3         example.to_numpy(),
4         lr.coef_.reshape(
5             2,
6         ),
7     )
8     + lr.intercept_
9 )

```

Out[41]: array([0.12151312])

This is the probability score of the positive class, which is USA.

In [42]:

```
1 lr.predict_proba([example])
```

Out[42]: array([[0.87848688, 0.12151312]])

With `predict_proba`, we get the same probability score for USA!!

- Let's visualize probability scores for some examples.

In [43]:

```

1 data_dict = {
2     "y": y_train.iloc[:12],
3     "y_hat": lr.predict(X_train.iloc[:12].to_numpy()).tolist(),
4     "probabilities": lr.predict_proba(X_train.iloc[:12].to_numpy()).tol
5 }

```

In [44]:

```
1 pd.DataFrame(data_dict)
```

Out[44]:

	y	y_hat	probabilities
160	Canada	Canada	[0.7046068097086481, 0.2953931902913519]
127	Canada	Canada	[0.563016906204013, 0.436983093795987]
169	Canada	Canada	[0.8389680973255864, 0.16103190267441364]
188	Canada	Canada	[0.7964150775404333, 0.20358492245956678]
187	Canada	Canada	[0.9010806652340972, 0.0989193347659027]
192	Canada	Canada	[0.7753006388010791, 0.2246993611989209]
62	USA	USA	[0.03074070460652778, 0.9692592953934722]
141	Canada	Canada	[0.6880304799160918, 0.3119695200839082]
183	Canada	Canada	[0.7891358587234145, 0.21086414127658554]
37	USA	USA	[0.006546969753885357, 0.9934530302461146]
50	USA	USA	[0.2787419584843098, 0.7212580415156902]
89	Canada	Canada	[0.8388877146644942, 0.1611122853355058]

The actual `y` and `y_hat` match in most of the cases but in some cases the model is more confident about the prediction than others.

Least confident cases

Let's examine some cases where the model is least confident about the prediction.

```
In [45]: 1 least_confident_X = X_train.loc[[127, 141]]
2 least_confident_X
```

Out[45]:

	longitude	latitude
127	-81.2496	42.9837
141	-79.6902	44.3893

```
In [46]: 1 least_confident_y = y_train.loc[[127, 141]]
2 least_confident_y
```

Out[46]:

127	Canada
141	Canada
Name:	country, dtype: object

```
In [47]: 1 probs = lr.predict_proba(least_confident_X.to_numpy())
2
3 data_dict = {
4     "y": least_confident_y,
5     "y_hat": lr.predict(least_confident_X.to_numpy()).tolist(),
6     "probability score (Canada)": probs[:, 0],
7     "probability score (USA)": probs[:, 1],
8 }
9 pd.DataFrame(data_dict)
```

Out[47]:

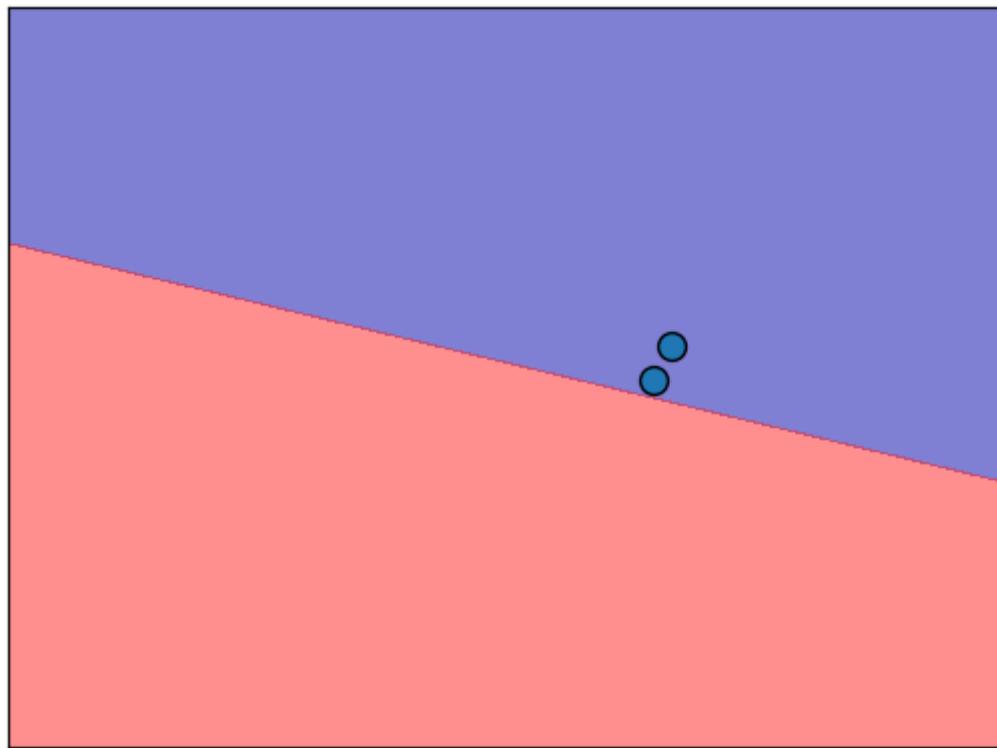
	y	y_hat	probability score (Canada)	probability score (USA)
127	Canada	Canada	0.563017	0.436983
141	Canada	Canada	0.688030	0.311970

In [48]:

```

1 mglearn.discrete_scatter(
2     least_confident_X.iloc[:, 0],
3     least_confident_X.iloc[:, 1],
4     least_confident_y,
5     markers="o",
6 )
7 mglearn.plots.plot_2d_separator(lr, X_train.to_numpy(), fill=True, eps=

```



The points are close to the decision boundary which makes sense.

Most confident cases

Let's examine some cases where the model is most confident about the prediction.

In [49]:

```

1 most_confident_X = X_train.loc[[37, 165]]
2 most_confident_X

```

Out[49]:

	longitude	latitude
37	-98.4951	29.4246
165	-52.7151	47.5617

In [50]:

```

1 most_confident_y = y_train.loc[[37, 165]]
2 most_confident_y

```

Out[50]:

37	USA
165	Canada
Name:	country, dtype: object

In [51]:

```

1 probs = lr.predict_proba(most_confident_X.to_numpy())
2
3 data_dict = {
4     "y": most_confident_y,
5     "y_hat": lr.predict(most_confident_X.to_numpy()).tolist(),
6     "probability score (Canada)": probs[:, 0],
7     "probability score (USA)": probs[:, 1],
8 }
9 pd.DataFrame(data_dict)

```

Out[51]:

	y	y_hat	probability score (Canada)	probability score (USA)
37	USA	USA	0.006547	0.993453
165	Canada	Canada	0.951092	0.048908

In [52]:

```
1 most_confident_X
```

Out[52]:

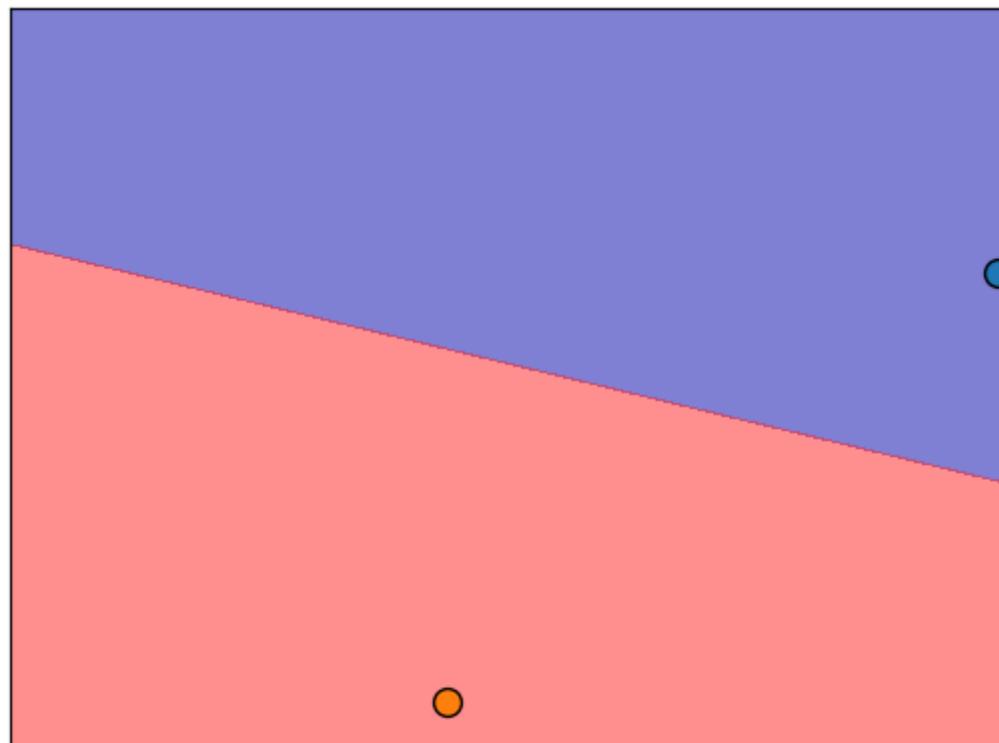
	longitude	latitude
37	-98.4951	29.4246
165	-52.7151	47.5617

In [53]:

```

1 mglearn.discrete_scatter(
2     most_confident_X.iloc[:, 0],
3     most_confident_X.iloc[:, 1],
4     most_confident_y,
5     markers="o",
6 )
7 mglearn.plots.plot_2d_separator(lr, X_train.to_numpy(), fill=True, eps=

```



The points are far away from the decision boundary which makes sense.

Over confident cases

Let's examine some cases where the model is confident about the prediction but the prediction is wrong.

```
In [54]: 1 over_confident_X = X_train.loc[[0, 1]]
2 over_confident_X
```

```
Out[54]:   longitude  latitude
0      -130.0437  55.9773
1     -134.4197  58.3019
```

```
In [55]: 1 over_confident_y = y_train.loc[[0, 1]]
2 over_confident_y
```

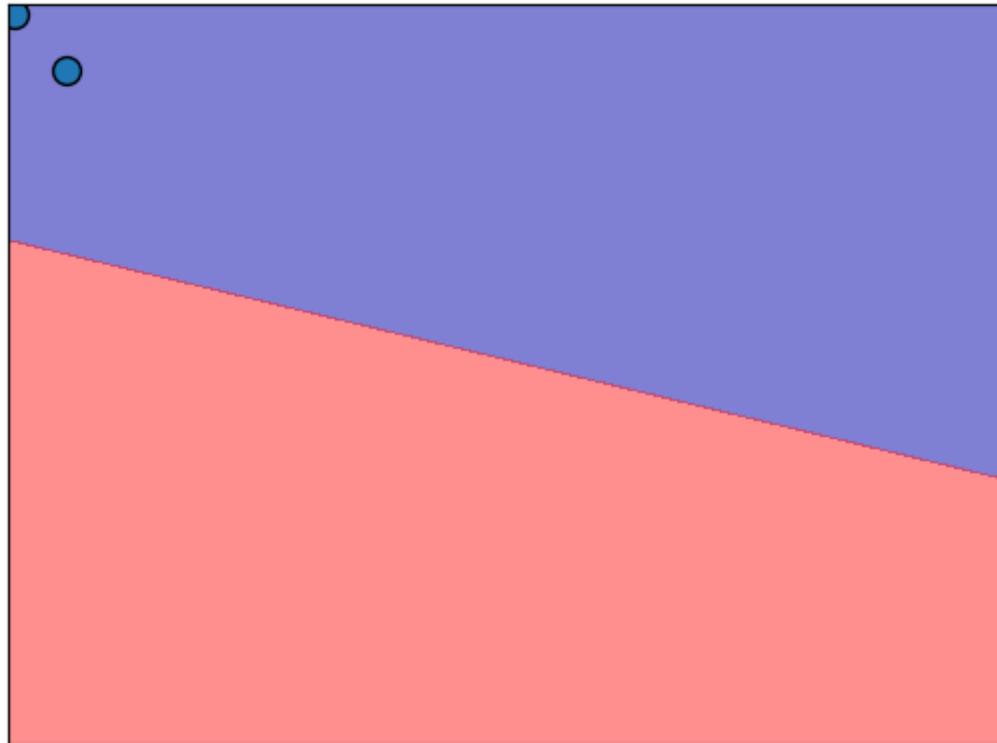
```
Out[55]: 0    USA
1    USA
Name: country, dtype: object
```

```
In [56]: 1 probs = lr.predict_proba(over_confident_X.to_numpy())
2
3 data_dict = {
4     "y": over_confident_y,
5     "y_hat": lr.predict(over_confident_X.to_numpy()).tolist(),
6     "probability score (Canada)": probs[:, 0],
7     "probability score (USA)": probs[:, 1],
8 }
9 pd.DataFrame(data_dict)
```

```
Out[56]:    y    y_hat  probability score (Canada)  probability score (USA)
0  USA    Canada            0.932487           0.067513
1  USA    Canada            0.961902           0.038098
```

In [57]:

```
1 mglearn.discrete_scatter(
2     over_confident_X.iloc[:, 0],
3     over_confident_X.iloc[:, 1],
4     over_confident_y,
5     markers="o",
6 )
7 mglearn.plots.plot_2d_separator(lr, X_train.to_numpy(), fill=True, eps=
```



- The cities are far away from the decision boundary. So the model is pretty confident about the prediction.
- But the cities are likely to be from Alaska and our linear model is not able to capture that this part belongs to the USA and not Canada.

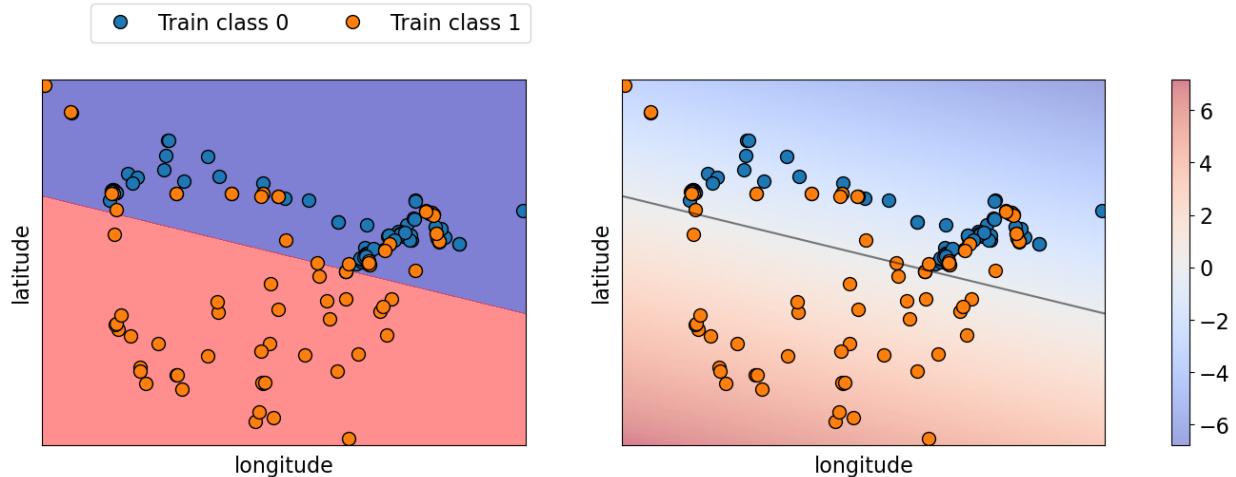
Below we are using colour to represent prediction probabilities. If you are closer to the border, the model is less confident whereas the model is more confident about the mainland cities, which makes sense.

In [58]:

```

1 fig, axes = plt.subplots(1, 2, figsize=(18, 5))
2 from matplotlib.colors import ListedColormap
3
4 for ax in axes:
5     mglearn.discrete_scatter(
6         X_train.iloc[:, 0], X_train.iloc[:, 1], y_train, markers="o", a
7     )
8     ax.set_xlabel("longitude")
9     ax.set_ylabel("latitude")
10
11 axes[0].legend(["Train class 0", "Train class 1"], ncol=2, loc=(0.1, 1.
12
13 mglearn.plots.plot_2d_separator(
14     lr, X_train.to_numpy(), fill=True, eps=0.5, ax=axes[0], alpha=0.5
15 )
16 mglearn.plots.plot_2d_separator(
17     lr, X_train.to_numpy(), fill=False, eps=0.5, ax=axes[1], alpha=0.5
18 )
19 scores_image = mglearn.tools.plot_2d_scores(
20     lr, X_train.to_numpy(), eps=0.5, ax=axes[1], alpha=0.5, cm=plt.cm.c
21 )
22 cbar = plt.colorbar(scores_image, ax=axes.tolist())

```



Sometimes a complex model that is overfitted, tends to make more confident predictions, even if they are wrong, whereas a simpler model tends to make predictions with more uncertainty.

To summarize,

- With hard predictions, we only know the class.
- With probability scores we know how confident the model is with certain predictions, which can be useful in understanding the model better.

? ? Questions for you

True/False (for practice)

- Increasing logistic regression's `C` hyperparameter increases model complexity.
- Unlike with Ridge regression, coefficients are not interpretable with logistic regression.
- The raw output score can be used to calculate the probability score for a given prediction.
- For linear classifier trained on d features, the decision boundary is a $d - 1$ -dimensional hyperplane.
- A linear model is likely to be uncertain about the data points close to the decision boundary.
- Similar to decision trees, conceptually logistic regression should be able to work with categorical features.
- Scaling might be a good idea in the context of logistic regression.

Zoom poll

1. The intercept of a linear model has a meaning similar to the model's coefficients (T/F)
2. The snake weight linear model was trained on snakes between 0 and 200 cm of length. Should we try predicting the weight of a 250 cm long snake?
3. A positive coefficient indicates positive correlation between the feature and the target (T/F)
4. RIDGE regression helps controlling the model complexity by limiting the number of non-zero coefficients in the model (T/F)
5. Logistic regression only accepts binary inputs (T/F)
6. Changing the order of the classes in logistic regression will not affect the coefficients (T/F)
7. A larger coefficient indicates a stronger correlation (T/F)

Linear SVM

- We have seen non-linear SVM with RBF kernel before. This is the default SVC model in `sklearn` because it tends to work better in many cases.
- There is also a linear SVM. You can pass `kernel="linear"` to create a linear SVM.

In [59]:

```

1 cities_df = pd.read_csv("../data/canada_usa_cities.csv")
2 train_df, test_df = train_test_split(cities_df, test_size=0.2, random_s
3 X_train, y_train = train_df.drop(columns=["country"], axis=1), train_df
4 X_test, y_test = test_df.drop(columns=["country"], axis=1), test_df["co

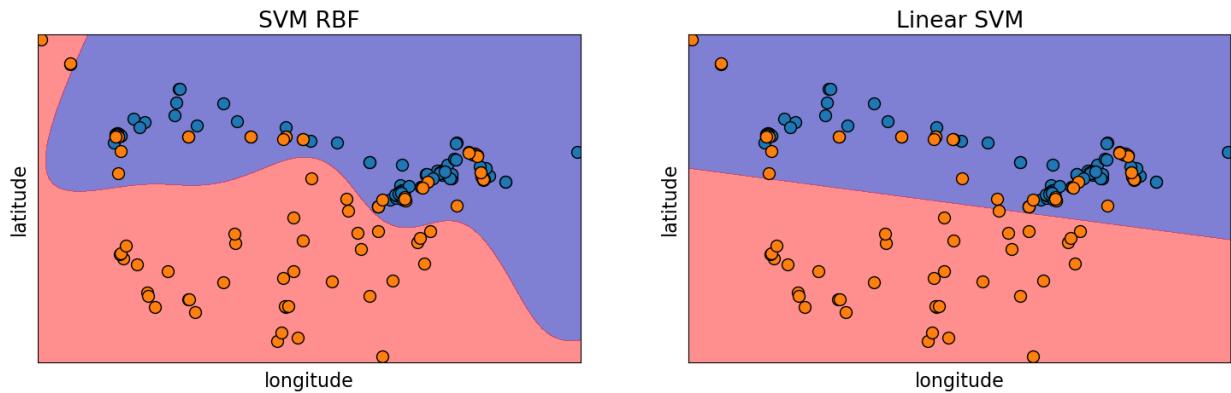
```

In [60]:

```

1 fig, axes = plt.subplots(1, 2, figsize=(18, 5))
2 from matplotlib.colors import ListedColormap
3
4 for (model, ax) in zip([SVC(gamma=0.01), SVC(kernel="linear")], axes):
5     mglearn.discrete_scatter(
6         X_train.iloc[:, 0].to_numpy(), X_train.iloc[:, 1].to_numpy(), y_train)
7     model.fit(X_train.to_numpy(), y_train)
8     ax.set_xlabel("longitude")
9     ax.set_ylabel("latitude")
10    mglearn.plots.plot_2d_separator(
11        model, X_train.to_numpy(), fill=True, eps=0.5, ax=ax, alpha=0.5
12    )
13
14
15 axes[0].set_title("SVM RBF")
16 axes[1].set_title("Linear SVM");

```



- `predict` method of linear SVM and logistic regression works the same way.
- We can get `coef_` associated with the features and `intercept_` using a Linear SVM model.

In [61]:

```

1 linear_svc = SVC(kernel="linear")
2 linear_svc.fit(X_train, y_train)
3 print("Model weights: %s" % (linear_svc.coef_))
4 print("Model intercept: %s" % (linear_svc.intercept_))

```

Model weights: [[-0.0195598 -0.23640124]]
 Model intercept: [8.22811601]

In [62]:

```

1 lr = LogisticRegression()
2 lr.fit(X_train, y_train)
3 print("Model weights: %s" % (lr.coef_))
4 print("Model intercept: %s" % (lr.intercept_))

```

Model weights: [[-0.04108149 -0.33683126]]
 Model intercept: [10.8869838]

- Note that the coefficients and intercept are slightly different for logistic regression.
- This is because the `fit` for linear SVM and logistic regression are different.

Model interpretation of linear classifiers

- One of the primary advantage of linear classifiers is their ability to interpret models.
- For example, with the sign and magnitude of learned coefficients we could answer questions such as which features are driving the prediction to which direction.
- We'll demonstrate this by training `LogisticRegression` on the famous [IMDB movie review \(<https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews>\)](https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews) dataset. The dataset is a bit large for demonstration purposes. So I am going to put a big portion of it in the test split to speed things up.

```
In [63]: 1 imdb_df = pd.read_csv("../data/imdb_master.csv", encoding="ISO-8859-1")
2 imdb_df = imdb_df[imdb_df["label"].str.startswith(("pos", "neg"))]
3 imdb_df.drop(["Unnamed: 0", "type", "file"], axis=1, inplace=True)
4 imdb_df.head()
```

Out[63]:

		review	label
0	Once again Mr. Costner has dragged out a movie for far longer than necessary. Aside from the terrific sea rescue sequences, of which there are very few I just did not care about any of the charact...		neg
1	This is an example of why the majority of action films are the same. Generic and boring, there's really nothing worth watching here. A complete waste of the then barely-tapped talents of Ice-T and...		neg
2	First of all I hate those moronic rappers, who could'nt act if they had a gun pressed against their foreheads. All they do is curse and shoot each other and acting like clichÃ©e version of gangst...		neg
3	Not even the Beatles could write songs everyone liked, and although Walter Hill is no mop-top he's second to none when it comes to thought provoking action movies. The nineties came and social pla...		neg
4	Brass pictures (movies is not a fitting word for them) really are somewhat brassy. Their alluring visual qualities are reminiscent of expensive high class TV commercials. But unfortunately Brass p...		neg

Let's clean up the data a bit.

```
In [64]: 1 import re
2
3
4 def replace_tags(doc):
5     doc = doc.replace("<br />", " ")
6     doc = re.sub("https://\S*", "", doc)
7     return doc
```

```
In [65]: 1 imdb_df["review_pp"] = imdb_df["review"].apply(replace_tags)
```

Are we breaking the Golden rule here?

Let's split the data and create bag of words representation.

```
In [66]: 1 train_df, test_df = train_test_split(imdb_df, test_size=0.9, random_state=42)
2 X_train, y_train = train_df["review_pp"], train_df["label"]
3 X_test, y_test = test_df["review_pp"], test_df["label"]
4 train_df.shape
```

Out[66]: (5000, 3)

```
In [67]: 1 vec = CountVectorizer(stop_words="english", max_features=10000)
2 bow = vec.fit_transform(X_train)
3 bow
```

Out[67]: <5000x10000 sparse matrix of type '<class 'numpy.int64'>'
with 383702 stored elements in Compressed Sparse Row format>

Examining the vocabulary

- The vocabulary (mapping from feature indices to actual words) can be obtained using `get_feature_names()` on the `CountVectorizer` object.

```
In [68]: 1 vocab = vec.get_feature_names_out()
```

```
In [69]: 1 vocab[0:10] # first few words
```

Out[69]: array(['00', '000', '01', '10', '100', '1000', '101', '11', '12', '13'],
dtype=object)

```
In [70]: 1 vocab[2000:2010] # some middle words
```

Out[70]: array(['conrad', 'cons', 'conscience', 'conscious', 'consciously',
'consciousness', 'consequence', 'consequences', 'conservative',
'conservatory'], dtype=object)

```
In [71]: 1 vocab[::-500] # words with a step of 500
```

Out[71]: array(['00', 'announcement', 'bird', 'cell', 'conrad', 'depth', 'elite',
'finnish', 'grimy', 'illusions', 'kerr', 'maltin', 'narrates',
'patients', 'publicity', 'reynolds', 'sfx', 'starting', 'thats',
'vance'], dtype=object)

Model building on the dataset

First let's try `DummyClassifier` on the dataset.

```
In [72]: 1 dummy = DummyClassifier()
2 scores = cross_validate(dummy, X_train, y_train, return_train_score=True)
3 pd.DataFrame(scores)
```

Out[72]:

	fit_time	score_time	test_score	train_score
0	0.002985	0.002214	0.505	0.505
1	0.002587	0.001751	0.505	0.505
2	0.002568	0.001805	0.505	0.505
3	0.004138	0.002139	0.505	0.505
4	0.003779	0.001940	0.505	0.505

We have a balanced dataset. So the `DummyClassifier` score is around 0.5.

Now let's try logistic regression.

```
In [73]: 1 pipe_lr = make_pipeline(
2     CountVectorizer(stop_words="english", max_features=10000),
3     LogisticRegression(max_iter=1000),
4 )
5 scores = cross_validate(pipe_lr, X_train, y_train, return_train_score=True)
6 pd.DataFrame(scores)
```

Out[73]:

	fit_time	score_time	test_score	train_score
0	1.354910	0.242920	0.847	1.0
1	1.171952	0.246972	0.832	1.0
2	1.238927	0.225187	0.842	1.0
3	1.142833	0.219543	0.853	1.0
4	1.294660	0.213265	0.839	1.0

Seems like we are overfitting. Let's optimize the hyperparameter `C`.

In [74]:

```

1 scores_dict = {
2     "C": 10.0 ** np.arange(-3, 3, 1),
3     "mean_train_scores": list(),
4     "mean_cv_scores": list(),
5 }
6 for C in scores_dict["C"]:
7     pipe_lr = make_pipeline(
8         CountVectorizer(stop_words="english", max_features=10000),
9         LogisticRegression(max_iter=1000, C=C),
10    )
11    scores = cross_validate(pipe_lr, X_train, y_train, return_train_score=True)
12    scores_dict["mean_train_scores"].append(scores["train_score"].mean())
13    scores_dict["mean_cv_scores"].append(scores["test_score"].mean())
14
15 results_df = pd.DataFrame(scores_dict)
16 results_df

```

Out[74]:

	C	mean_train_scores	mean_cv_scores
0	0.001	0.83470	0.7964
1	0.010	0.92265	0.8456
2	0.100	0.98585	0.8520
3	1.000	1.00000	0.8426
4	10.000	1.00000	0.8376
5	100.000	1.00000	0.8350

In [75]:

```

1 optimized_C = results_df["C"].iloc[np.argmax(results_df["mean_cv_scores"])
2 print(
3     "The maximum validation score is %.3f at C = %.2f "
4     % (
5         np.max(results_df["mean_cv_scores"]),
6         optimized_C,
7     )
8 )

```

The maximum validation score is 0.852 at C = 0.10

Let's train a model on the full training set with the optimized hyperparameter values.

```
In [76]: 1 pipe_lr = make_pipeline(
2     CountVectorizer(stop_words="english", max_features=10000),
3     LogisticRegression(max_iter=1000, C=optimized_C),
4 )
5 pipe_lr.fit(X_train, y_train)
```

```
Out[76]: Pipeline(steps=[('countvectorizer',
                           CountVectorizer(max_features=10000, stop_words='english')),
                           ('logisticregression',
                           LogisticRegression(C=0.1, max_iter=1000))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Examining learned coefficients

- The learned coefficients are exposed by the `coef_` attribute of [LogisticRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) object.

```
In [77]: 1 feature_names = np.array(pipe_lr.named_steps["countvectorizer"].get_feature_names())
2 coeffs = pipe_lr.named_steps["logisticregression"].coef_.flatten()
```

```
In [78]: 1 word_coeff_df = pd.DataFrame(coeffs, index=feature_names, columns=["Coefficient"])
2 word_coeff_df
```

Out[78]:

	Coefficient
00	-0.074949
000	-0.083893
01	-0.034402
10	0.056493
100	0.041633
...	...
zoom	-0.013299
zooms	-0.022139
zorak	0.021878
zorro	0.130075
â½	0.012649

10000 rows × 1 columns

- Let's sort the coefficients in descending order.
- Interpretation
 - if $w_j > 0$ then increasing x_{ij} moves us toward predicting +1.
 - if $w_j < 0$ then increasing x_{ij} moves us toward predicting -1.

In [79]: 1 word_coeff_df.sort_values(by="Coefficient", ascending=False)

Out[79]:

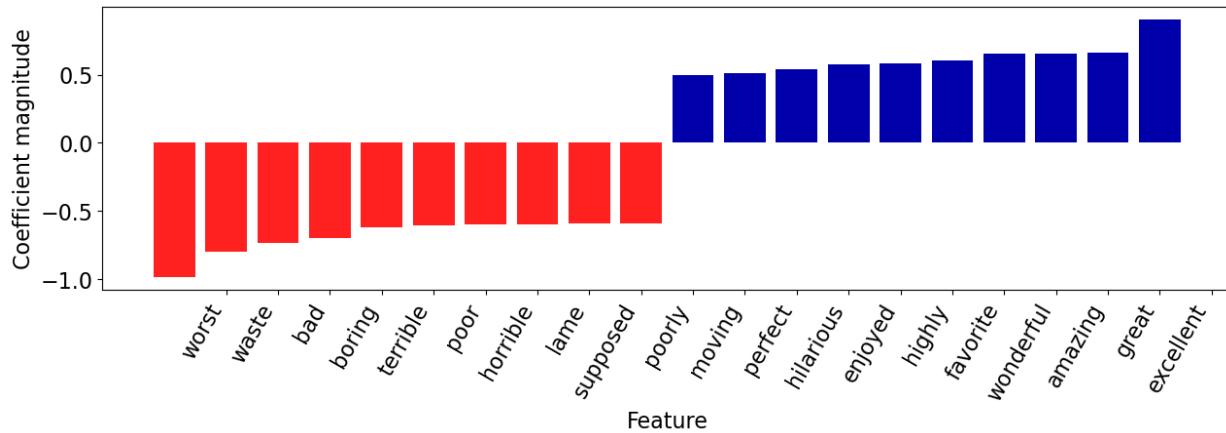
	Coefficient
excellent	0.903484
great	0.659922
amazing	0.653301
wonderful	0.651763
favorite	0.607887
...	...
terrible	-0.621695
boring	-0.701030
bad	-0.736608
waste	-0.799353
worst	-0.986970

10000 rows × 1 columns

- The coefficients make sense!

Let's visualize the top 10 features.

In [80]: 1 mglearn.tools.visualize_coefficients(coeffs, feature_names, n_top_featu



Let's explore prediction of the following new review.

```
In [81]: 1 fake_review = "It got a bit boring at times but the direction was excel
```

```
In [82]: 1 feat_vec = pipe_lr.named_steps["countvectorizer"].transform([fake_revie
```

```
In [83]: 1 feat_vec
```

```
Out[83]: <1x10000 sparse matrix of type '<class 'numpy.int64'>'  
with 12 stored elements in Compressed Sparse Row format>
```

Let's get prediction probability scores of the fake review.

```
In [84]: 1 pipe_lr.predict_proba([fake_review])
```

```
Out[84]: array([[0.1718113, 0.8281887]])
```

The model is 82% confident that it's a positive review.

```
In [85]: 1 pipe_lr.predict([fake_review])[0]
```

```
Out[85]: 'pos'
```

We can find which of the vocabulary words are present in this review:

```
In [86]: 1 feat_vec.toarray().ravel().astype(bool)
```

```
Out[86]: array([False, False, False, ..., False, False, False])
```

```
In [87]: 1 words_in_ex = feat_vec.toarray().ravel().astype(bool)  
2 words_in_ex
```

```
Out[87]: array([False, False, False, ..., False, False, False])
```

How many of the words are in this review?

```
In [88]: 1 np.sum(words_in_ex)
```

```
Out[88]: 12
```

```
In [89]: 1 np.array(feature_names)[words_in_ex]
```

```
Out[89]: array(['acting', 'bit', 'boring', 'direction', 'enjoyed', 'excellent',  
'flawless', 'got', 'highly', 'movie', 'overall', 'times'],  
dtype=object)
```

In [90]:

```

1 ex_df = pd.DataFrame(
2     data=coeffs[words_in_ex],
3     index=np.array(feature_names)[words_in_ex],
4     columns=[ "Coefficient" ],
5 )
6 ex_df

```

Out[90]:

	Coefficient
acting	-0.126498
bit	0.390053
boring	-0.701030
direction	-0.268316
enjoyed	0.578879
excellent	0.903484
flawless	0.113743
got	-0.122759
highly	0.582012
movie	-0.037942
overall	0.136288
times	0.133895

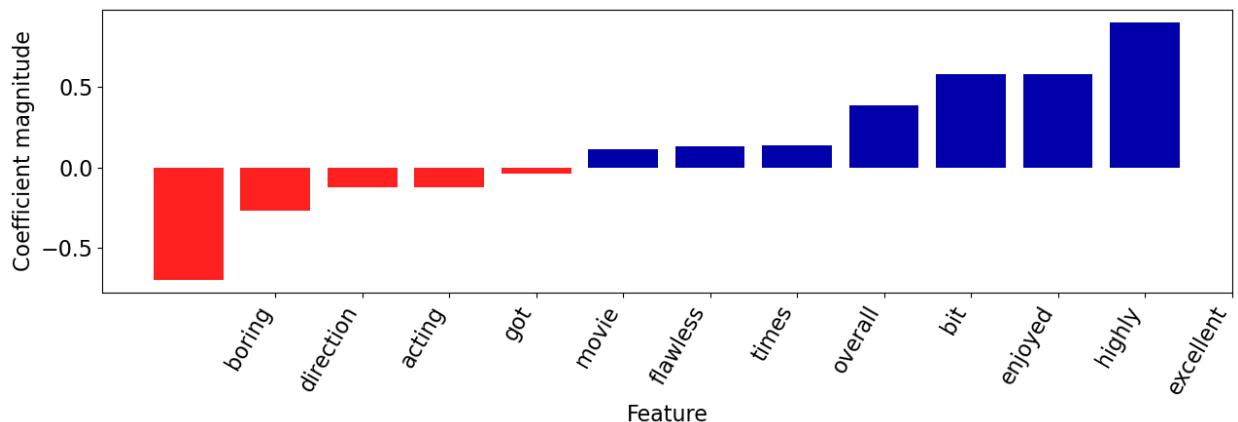
Let's visualize how the words with positive and negative coefficients are driving the hard prediction.

In [91]:

```

1 mglearn.tools.visualize_coefficients(
2     coeffs[words_in_ex], np.array(feature_names)[words_in_ex], n_top_fe
3 )

```



In [92]:

```

1 def plot_coeff_example(feat_vect, coeffs, feature_names):
2     words_in_ex = feat_vec.toarray().ravel().astype(bool)
3
4     ex_df = pd.DataFrame(
5         data=coeffs[words_in_ex],
6         index=np.array(feature_names)[words_in_ex],
7         columns=[ "Coefficient" ],
8     )
9     return ex_df

```

Most positive review

- Remember that you can look at the probabilities (confidence) of the classifier's prediction using the `model.predict_proba` method.
- Can we find the messages where our classifier is most confident or least confident?

In [93]:

```

1 pos_probs = pipe_lr.predict_proba(X_train)[
2     :, 1
3 ] # only get probabilities associated with pos class
4 pos_probs

```

Out[93]: array([0.95205899, 0.83301769, 0.9093526 , ..., 0.89247531, 0.05736279, 0.79360853])

Let's get the index of the example where the classifier is most confident (highest `predict_proba` score for positive).

In [94]:

```
1 most_positive = np.argmax(pos_probs)
```

In [95]: 1 X_train.iloc[most_positive]

Out[95]: 'Moving beyond words is this heart breaking story of a divorce which results in a tragic custody battle over a seven year old boy. One of "Kramer v. Kramer\"s" great strengths is its screenwriter director Robert Benton, who has marvellously adapted Avery Corman\\'s novel to the big screen. He keeps things beautifully simple and most realistic, while delivering all the drama straight from the heart. His talent for telling emotional tales like this was to prove itself again with "Places in the Heart", where he showed, as in "Kramer v. Kramer", that he has a natural ability for working with children. The picture\\'s other strong point is the splendid acting which deservedly received four of the film\\'s nine Academy Award nominations, two of them walking away winners. One of those was Dustin Hoffman (Best Actor), who is superb as frustrated business man Ted Kramer, a man who has forgotten that his wife is a person. As said wife Joanne, Meryl Streep claimed the supporting actress Oscar for a strong, sensitive portrayal of a woman who had lost herself in eight years of marriage. Also nominated was Jane Alexander for her fantastic turn as the Kramer\\'s good friend Margaret. Final word in the acting stakes must go to young Justin Henry, whose incredibly moving performance will find you choking back tears again and again, and a thoroughly deserved Oscar nomination came his way. Brilliant also is Nestor Almendros\\' cinematography and Jerry Greenberg \\'s timely editing, while musically Henry Purcell\\'s classical piece is used to effect. Truly this is a touching story of how a father and son come to depend on each other when their wife and mother leaves. They grow together, come to know each other and form an entirely new and wonderful relationship. Ted finds himself with new responsibilities and a new outlook on life, and slowly comes to realise why Joanne had to go. Certainly if nothing else, "Kramer v. Kramer" demonstrates that nobody wins when it comes to a custody battle over a young child, especially not the child himself. Saturday, June 10, 1995 - T.V. Strong drama from Avery Corman\\'s novel about the heartache of a custody battle between estranged parents who both feel they have the child\\'s best interests at heart. Aside from a superb screenplay and amazingly controlled direction, both from Robert Benton, it\\'s the superlative cast that make this picture such a winner. Hoffman is brilliant as Ted Kramer, the man torn between his toppling career and the son whom he desperately wants to keep. Excellent too is Streep as the woman lost in eight years of marriage who had to get out before she faded to nothing as a person. In support of these two is a very strong Jane Alexander as mutual friend Margaret, an outstanding Justin Henry as the boy caught in the middle, and a top cast of extras. This highly emotional, heart rending drama more than deserved it\\'s 1979 Academy Awards for best film, best actor (Hoffman) and best supporting actress (Streep). Wednesday, February 28, 1996 - T.V.'

In [96]: 1 print("True target: %s\n" % (y_train.iloc[most_positive]))
2 print("Predicted target: %s\n" % (pipe_lr.predict(X_train.iloc[[most_pos]])))
3 print("Prediction probability: %.4f" % (pos_probs[most_positive]))

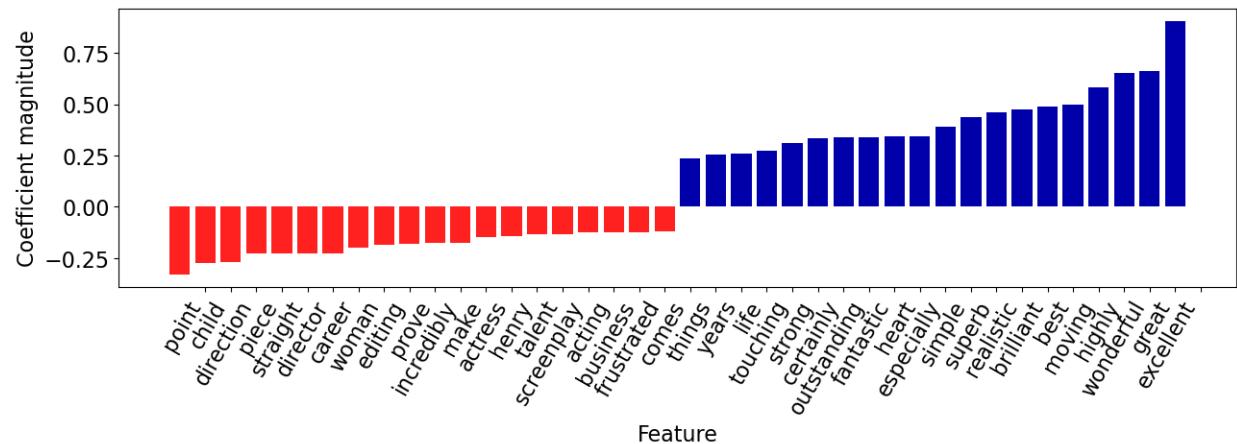
True target: pos

Predicted target: pos

Prediction probability: 1.0000

Let's examine the features associated with the review.

```
In [97]: 1 feat_vec = pipe_lr.named_steps["countvectorizer"].transform(
2     X_train.iloc[[most_positive]])
3 )
4 words_in_ex = feat_vec.toarray().ravel().astype(bool)
5 mglearn.tools.visualize_coefficients(
6     coeffs[words_in_ex], np.array(feature_names)[words_in_ex], n_top_fe
7 )
```



The review has both positive and negative words but the words with **positive** coefficients win in this case!

Most negative review

```
In [98]: 1 neg_probs = pipe_lr.predict_proba(X_train)[
2     :, 0
3 ] # only get probabilities associated with pos class
4 neg_probs
```

```
Out[98]: array([0.04794101, 0.16698231, 0.0906474 , ..., 0.10752469, 0.94263721,
0.20639147])
```

```
In [99]: 1 most_negative = np.argmax(neg_probs)
```

In [100]:

```

1 print("Review: %s\n" % (X_train.iloc[[most_negative]]))
2 print("True target: %s\n" % (y_train.iloc[most_negative]))
3 print("Predicted target: %s\n" % (pipe_lr.predict(X_train.iloc[[most_ne
4 print("Prediction probability: %0.4f" % (pos_probs[most_negative])))
```

Review: 36555 I made the big mistake of actually watching this whole movie a few nights ago. God I'm still trying to recover. This movie does not even deserve a 1.4 average. IMDb needs to have 0 vote ratings po...
Name: review_pp, dtype: object

True target: neg

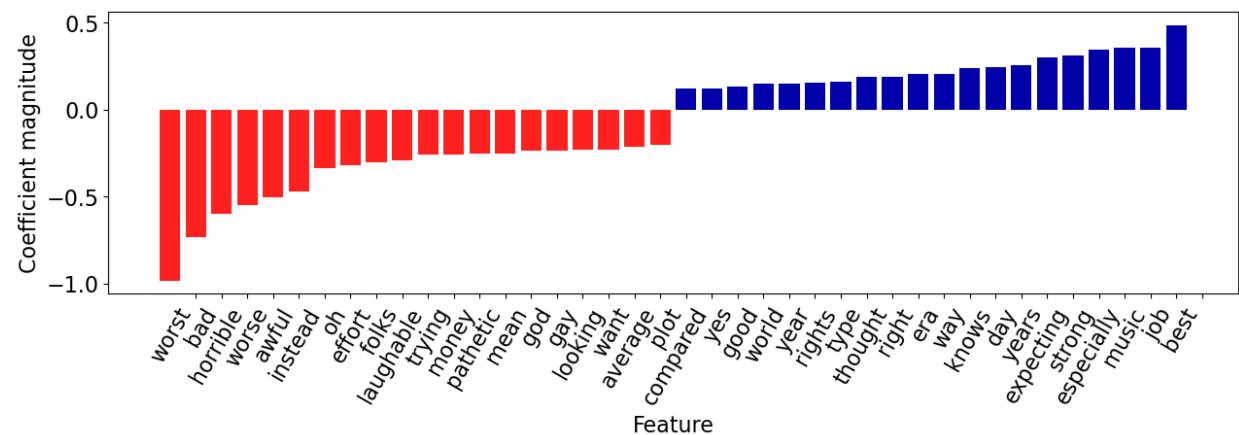
Predicted target: neg

Prediction probability: 0.0000

In [101]:

```

1 feat_vec = pipe_lr.named_steps["countvectorizer"].transform(
2     X_train.iloc[[most_negative]])
3 )
4 words_in_ex = feat_vec.toarray().ravel().astype(bool)
5 mglearn.tools.visualize_coefficients(
6     coeffs[words_in_ex], np.array(feature_names)[words_in_ex], n_top_fe
7 )
```



The review has both positive and negative words but the words with negative coefficients win in this case!

? ? Questions for you

Question for you to ponder on

- Is it possible to identify most important features using k -NNs? What about decision trees?

Summary of linear models

- Linear regression is a linear model for regression whereas logistic regression is a linear model for classification.
- Both these models learn one coefficient per feature, plus an intercept.

Main hyperparameters

- The main hyperparameter is the "regularization" hyperparameter controlling the fundamental tradeoff.
 - Logistic Regression: `C`
 - Linear SVM: `C`
 - Ridge: `alpha`

Interpretation of coefficients in linear models

- the j th coefficient tells us how feature j affects the prediction
- if $w_j > 0$ then increasing x_{ij} moves us toward predicting $+1$
- if $w_j < 0$ then increasing x_{ij} moves us toward prediction -1
- if $w_j == 0$ then the feature is not used in making a prediction

Strengths of linear models

- Fast to train and predict
- Scale to large datasets and work well with sparse data
- Relatively easy to understand and interpret the predictions
- Perform well when there is a large number of features

Limitations of linear models

- Is your data "linearly separable"? Can you draw a hyperplane between these datapoints that separates them with 0 error.
 - If the training examples can be separated by a linear decision rule, they are **linearly separable**.

A few questions you might be thinking about

- How often the real-life data is linearly separable?
- Is the following XOR function linearly separable?

x_1	x_2	target
0	0	0
0	1	1
1	0	1
1	1	0

CPSC 330

Applied Machine Learning

Lecture 8: Hyperparameter Optimization and Optimization Bias

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

In [79]:

```
1 import os
2 import sys
3
4 sys.path.append("../code/.")
5
6 import IPython
7 import ipywidgets as widgets
8 import matplotlib.pyplot as plt
9 import mglearn
10 import numpy as np
11 import pandas as pd
12 from IPython.display import HTML, display
13 from ipywidgets import interact, interactive
14 from plotting_functions import *
15 from sklearn.dummy import DummyClassifier
16 from sklearn.feature_extraction.text import CountVectorizer, TfidfVector
17 from sklearn.impute import SimpleImputer
18 from sklearn.model_selection import cross_val_score, cross_validate, tr
19 from sklearn.pipeline import Pipeline, make_pipeline
20 from sklearn.preprocessing import OneHotEncoder, StandardScaler
21 from sklearn.svm import SVC
22 from sklearn.tree import DecisionTreeClassifier
23 from utils import *
24
25 %matplotlib inline
26 pd.set_option("display.max_colwidth", 200)
```

Learning outcomes

From this lecture, you will be able to

- explain the need for hyperparameter optimization
- carry out hyperparameter optimization using `sklearn`'s `GridSearchCV` and `RandomizedSearchCV`
- explain different hyperparameters of `GridSearchCV`
- explain the importance of selecting a good range for the values.
- explain optimization bias
- identify and reason when to trust and not trust reported accuracies

Hyperparameter optimization motivation

Motivation

- Remember that the fundamental goal of supervised machine learning is to generalize beyond what we see in the training examples.
- We have been using data splitting and cross-validation to provide a framework to approximate generalization error.
- With this framework, we can improve the model's generalization performance by tuning model hyperparameters using cross-validation on the training set.

Hyperparameters: the problem

- In order to improve the generalization performance, finding the best values for the important hyperparameters of a model is necessary for almost all models and datasets.
- Picking good hyperparameters is important because if we don't do it, we might end up with an underfit or overfit model.

Some ways to pick hyperparameters:

- Manual or expert knowledge or heuristics based optimization
- Data-driven or automated optimization

Manual hyperparameter optimization

- Advantage: we may have some intuition about what might work.
 - E.g. if I'm massively overfitting, try decreasing `max_depth` or `C`.
- Disadvantages
 - it takes a lot of work
 - not reproducible
 - in very complicated cases, our intuition might be worse than a data-driven approach

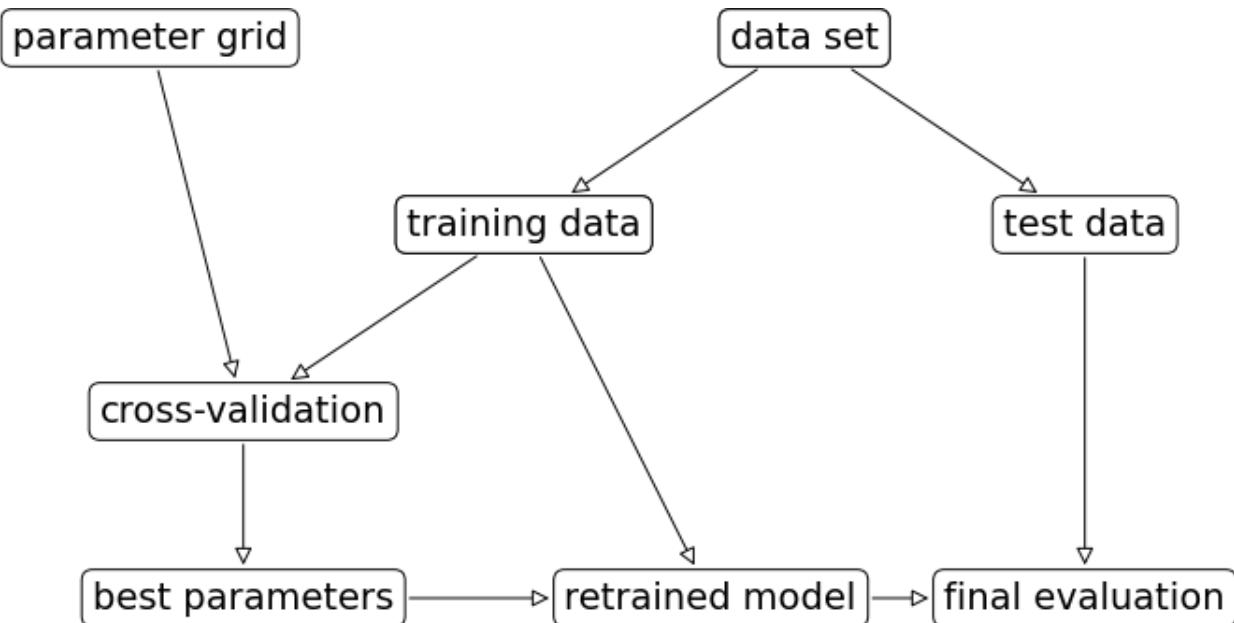
Automated hyperparameter optimization

- Formulate the hyperparameter optimization as one big search problem.
- Often we have many hyperparameters of different types: Categorical, integer, and continuous.
- Often, the search space is quite big and systematic search for optimal values is infeasible.

In homework assignments, we have been carrying out hyperparameter search by exhaustively trying different possible combinations of the hyperparameters of interest.

This is what it looks like schematically:

In [80]: 1 `mglearn.plots.plot_grid_search_overview()`



Let's look at an example of tuning `max_depth` of the `DecisionTreeClassifier` on the Spotify dataset.

```
In [81]: 1 spotify_df = pd.read_csv("../data/spotify.csv", index_col=0)
2 X_spotify = spotify_df.drop(columns=["target", "song_title", "artist"])
3 y_spotify = spotify_df["target"]
4 X_spotify.head()
```

Out[81]:

	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode
0	0.0102	0.833	204600	0.434	0.021900	2	0.1650	-8.795	1
1	0.1990	0.743	326933	0.359	0.006110	1	0.1370	-10.401	1
2	0.0344	0.838	185707	0.412	0.000234	2	0.1590	-7.148	1
3	0.6040	0.494	199413	0.338	0.510000	5	0.0922	-15.236	1
4	0.1800	0.678	392893	0.561	0.512000	5	0.4390	-11.648	0

```
In [82]: 1 X_train, X_test, y_train, y_test = train_test_split(
2     X_spotify, y_spotify, test_size=0.2, random_state=123
3 )
```

```
In [83]: 1 best_score = 0
2
3 param_grid = {"max_depth": np.arange(1, 20, 2)}
4
5 results_dict = {"max_depth": [], "mean_cv_score": []}
6
7 for depth in param_grid[
8     "max_depth"
9 ]: # for each combination of parameters, train an SVC
10    dt = DecisionTreeClassifier(max_depth=depth)
11    scores = cross_val_score(dt, X_train, y_train) # perform cross-val
12    mean_score = np.mean(scores) # compute mean cross-validation accuracy
13    if (
14        mean_score > best_score
15    ): # if we got a better score, store the score and parameters
16        best_score = mean_score
17        best_params = {"max_depth": depth}
18    results_dict["max_depth"].append(depth)
19    results_dict["mean_cv_score"].append(mean_score)
```

```
In [84]: 1 best_params
```

Out[84]: {'max_depth': 5}

```
In [85]: 1 best_score
```

Out[85]: 0.7191604330519393

Let's try SVM RBF and tuning C and gamma on the same dataset.

```
In [86]: 1 pipe_svm = make_pipeline(StandardScaler(), SVC()) # We need scaling for
          2 pipe_svm.fit(X_train, y_train)
```

Out[86]: Pipeline(steps=[('standardscaler', StandardScaler()), ('svc', SVC())])
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Let's try cross-validation with default hyperparameters of SVC.

```
In [87]: 1 scores = cross_validate(pipe_svm, X_train, y_train, return_train_score=True)
          2 pd.DataFrame(scores).mean()
```

Out[87]: fit_time 0.074381
score_time 0.032224
test_score 0.738998
train_score 0.814011
dtype: float64

Now let's try exhaustive hyperparameter search using for loops.

This is what we have been doing for this:

```
for gamma in [0.01, 1, 10, 100]: # for some values of gamma
    for C in [0.01, 1, 10, 100]: # for some values of C
        for fold in folds:
            fit in training portion with the given C
            score on validation portion
            compute average score

pick hyperparameter values which yield with best average score
```

```
In [88]: 1 best_score = 0
2
3 param_grid = {
4     "C": [0.001, 0.01, 0.1, 1, 10, 100],
5     "gamma": [0.001, 0.01, 0.1, 1, 10, 100],
6 }
7
8 results_dict = {"C": [], "gamma": [], "mean_cv_score": []}
9
10 for gamma in param_grid["gamma"]:
11     for C in param_grid["C"]:
12         pipe_svm = make_pipeline(StandardScaler(), SVC(gamma=gamma, C=C))
13         scores = cross_val_score(pipe_svm, X_train, y_train) # perform
14         mean_score = np.mean(scores) # compute mean cross-validation a
15         if (
16             mean_score > best_score
17         ): # if we got a better score, store the score and parameters
18             best_score = mean_score
19             best_parameters = {"C": C, "gamma": gamma}
20             results_dict["C"].append(C)
21             results_dict["gamma"].append(gamma)
22             results_dict["mean_cv_score"].append(mean_score)
```

```
In [89]: 1 best_parameters
```

```
Out[89]: {'C': 1, 'gamma': 0.1}
```

```
In [90]: 1 best_score
```

```
Out[90]: 0.7439609253312309
```

```
In [91]: 1 df = pd.DataFrame(results_dict)
```

```
In [92]: 1 df.sort_values(by="mean_cv_score", ascending=False).head(10)
```

```
Out[92]:
```

	C	gamma	mean_cv_score
15	1.0	0.100	0.743961
11	100.0	0.010	0.732792
16	10.0	0.100	0.729091
10	10.0	0.010	0.720391
17	100.0	0.100	0.711715
5	100.0	0.001	0.704284
14	0.1	0.100	0.703034
9	1.0	0.010	0.697473
8	0.1	0.010	0.678851
4	10.0	0.001	0.678244

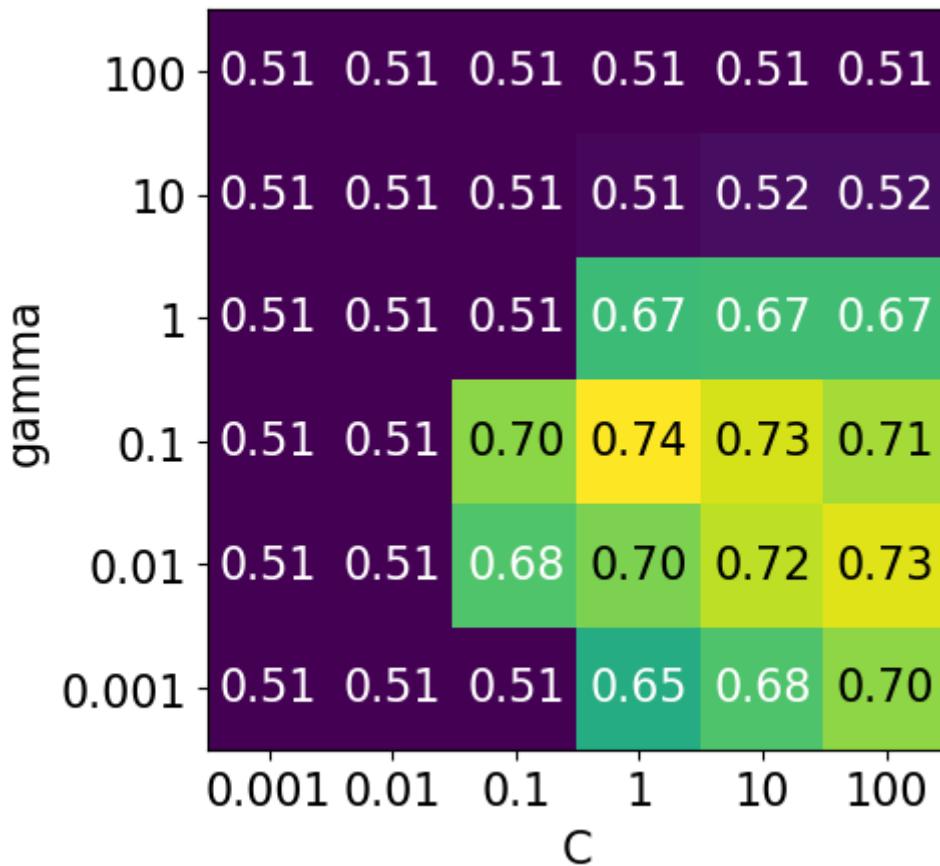
In [93]:

```

1 scores = np.array(df.mean_cv_score).reshape(6, 6)
2
3 # plot the mean cross-validation scores
4 mglearn.tools.heatmap(
5     scores,
6     xlabel="C",
7     xticklabels=param_grid["C"],
8     ylabel="gamma",
9     yticklabels=param_grid["gamma"],
10    cmap="viridis",
11 )

```

Out[93]: <matplotlib.collections.PolyCollection at 0x18dec81c0>



- We have 6 possible values for `C` and 6 possible values for `gamma`.
- In 5-fold cross-validation, for each combination of parameter values, five accuracies are computed.
- So to evaluate the accuracy of the SVM using 6 values of `C` and 6 values of `gamma` using five-fold cross-validation, we need to train $36 * 5 = 180$ models!

In [94]:

```
1 np.prod(list(map(len, param_grid.values())))
```

Out[94]: 36

Once we have optimized hyperparameters, we retrain a model on the full training set with these optimized hyperparameters.

```
In [95]: 1 pipe_svm = make_pipeline(StandardScaler(), SVC(**best_parameters))
2 pipe_svm.fit(
3     X_train, y_train
4 ) # Retrain a model with optimized hyperparameters on the combined tra
```

Out[95]: Pipeline(steps=[('standardscaler', StandardScaler()), ('svc', SVC(C=1, gamma=0.1))])

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

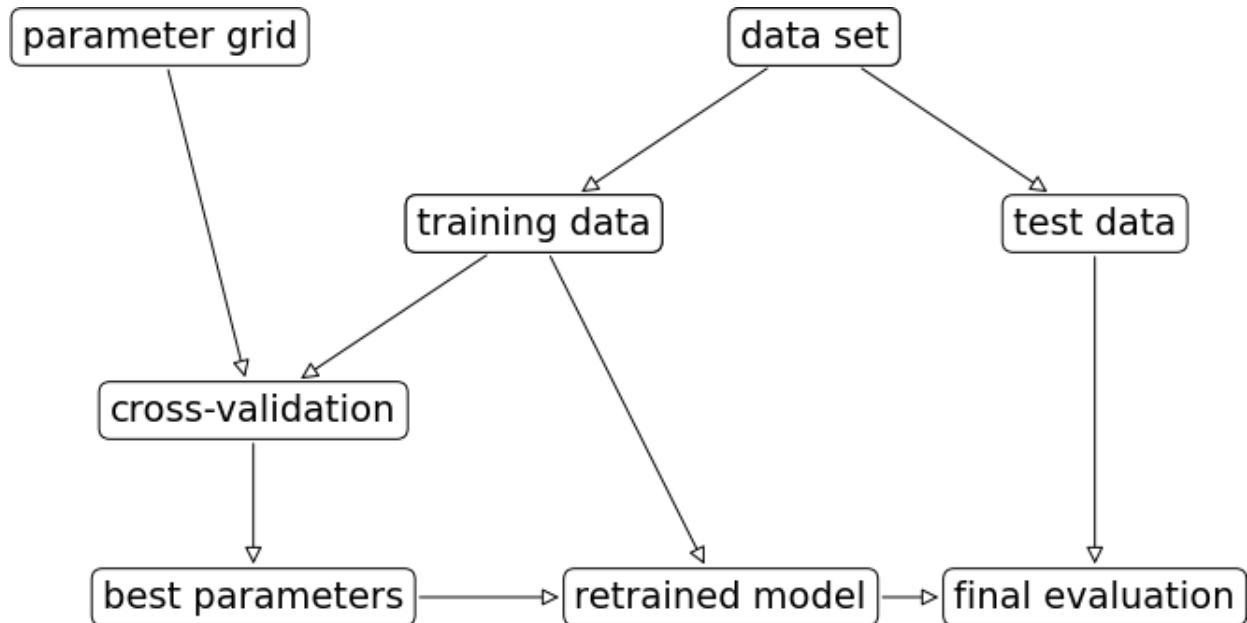
And finally evaluate the performance of this model on the test set.

```
In [96]: 1 pipe_svm.score(X_test, y_test) # Final evaluation on the test data
```

Out[96]: 0.7376237623762376

This process is so common that there are some standard methods in scikit-learn where we can carry out all of this in a more compact way.

```
In [97]: 1 mglearn.plots.plot_grid_search_overview()
```



In this lecture we are going to talk about two such most commonly used automated optimizations methods from scikit-learn .

- Exhaustive grid search: [sklearn.model_selection.GridSearchCV \(\[http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html\]\(http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html\)\)](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html).

- Randomized search: [`sklearn.model_selection.RandomizedSearchCV`](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html) (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html)

The "CV" stands for cross-validation; these methods have built-in cross-validation.

Exhaustive grid search: `sklearn.model_selection.GridSearchCV` http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

- For `GridSearchCV` we need
 - an instantiated model or a pipeline
 - a parameter grid: A user specifies a set of values for each hyperparameter.
 - other optional arguments

The method considers product of the sets and then evaluates each combination one by one.

In [98]:

```

1 from sklearn.model_selection import GridSearchCV
2
3 pipe_svm = make_pipeline(StandardScaler(), SVC())
4
5 param_grid = {
6     "svc__gamma": [0.001, 0.01, 0.1, 1.0, 10, 100],
7     "svc__C": [0.001, 0.01, 0.1, 1.0, 10, 100],
8 }
9
10 grid_search = GridSearchCV(
11     pipe_svm, param_grid, cv=5, n_jobs=-1, return_train_score=True
12 )

```

In [99]:

```

1 from sklearn import set_config
2
3 set_config(display="diagram")

```

The `GridSearchCV` object above behaves like a classifier. We can call `fit`, `predict` or `score` on it.

```
In [100]: 1 grid_search.fit(X_train, y_train) # all the work is done here  
2 grid_search
```

```
Out[100]: GridSearchCV(cv=5,  
                         estimator=Pipeline(steps=[('standardscaler', StandardScaler  
()),  
                                         ('svc', SVC())]),  
                         n_jobs=-1,  
                         param_grid={'svc__C': [0.001, 0.01, 0.1, 1.0, 10, 100],  
                                         'svc__gamma': [0.001, 0.01, 0.1, 1.0, 10, 100]},  
                         return_train_score=True)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Fitting the `GridSearchCV` object

- Searches for the best hyperparameter values
- You can access the best score and the best hyperparameters using `best_score_` and `best_params_` attributes, respectively.

```
In [101]: 1 grid_search.best_score_
```

```
Out[101]: 0.7439609253312309
```

```
In [102]: 1 grid_search.best_params_
```

```
Out[102]: {'svc__C': 1.0, 'svc__gamma': 0.1}
```

- It is often helpful to visualize results of all cross-validation experiments.
- You can access this information using `cv_results_` attribute of a fitted `GridSearchCV` object.

In [103]:

```

1 results = pd.DataFrame(grid_search.cv_results_)
2 results.T

```

Out[103]:

	0	1	2	3	4
mean_fit_time	0.233338	0.222731	0.227929	0.2085	0.20639
std_fit_time	0.001997	0.016612	0.01265	0.006013	0.008111
mean_score_time	0.127997	0.118951	0.117631	0.115487	0.118488
std_score_time	0.001649	0.015134	0.006908	0.003849	0.003969
param_svc_C	0.001	0.001	0.001	0.001	0.001
param_svc_gamma	0.001	0.01	0.1	1.0	10
params	{'svc_C': 0.001, 'svc_gamma': 0.001}, {'svc_C': 0.001, 'svc_gamma': 0.01}, {'svc_C': 0.001, 'svc_gamma': 0.1}, {'svc_C': 0.001, 'svc_gamma': 1.0}, {'svc_C': 0.001, 'svc_gamma': 10}				
split0_test_score	0.50774	0.50774	0.50774	0.50774	0.50774
split1_test_score	0.50774	0.50774	0.50774	0.50774	0.50774
split2_test_score	0.50774	0.50774	0.50774	0.50774	0.50774
split3_test_score	0.506211	0.506211	0.506211	0.506211	0.506211
split4_test_score	0.509317	0.509317	0.509317	0.509317	0.509317
mean_test_score	0.50775	0.50775	0.50775	0.50775	0.50775
std_test_score	0.000982	0.000982	0.000982	0.000982	0.000982
rank_test_score	21	21	21	21	21
split0_train_score	0.507752	0.507752	0.507752	0.507752	0.507752
split1_train_score	0.507752	0.507752	0.507752	0.507752	0.507752
split2_train_score	0.507752	0.507752	0.507752	0.507752	0.507752
split3_train_score	0.508133	0.508133	0.508133	0.508133	0.508133
split4_train_score	0.507359	0.507359	0.507359	0.507359	0.507359
mean_train_score	0.50775	0.50775	0.50775	0.50775	0.50775
std_train_score	0.000245	0.000245	0.000245	0.000245	0.000245

22 rows × 36 columns

In [104]:

```

1 results = (
2     pd.DataFrame(grid_search.cv_results_).set_index("rank_test_score").
3 )
4 results.T

```

Out[104]:

rank_test_score	1	2	3	4	5
mean_fit_time	0.171719	0.274006	0.234079	0.177106	0.46566
std_fit_time	0.005016	0.010851	0.011819	0.006139	0.025558
mean_score_time	0.084812	0.072378	0.076974	0.089491	0.072994
std_score_time	0.002488	0.002687	0.003899	0.006593	0.00529
param_svc_C	1.0	100	10	10	100
param_svc_gamma	0.1	0.01	0.1	0.01	0.1
params	{'svc_C': 1.0, 'svc_gamma': 0.1}	{'svc_C': 100, 'svc_gamma': 0.01}	{'svc_C': 10, 'svc_gamma': 0.1}	{'svc_C': 10, 'svc_gamma': 0.01}	{'svc_C': 100, 'svc_gamma': 0.1}
split0_test_score	0.755418	0.73065	0.702786	0.739938	0.705882
split1_test_score	0.755418	0.758514	0.767802	0.733746	0.76161
split2 test score	0.712074	0.71517	0.693498	0.696594	0.671827

Let's only look at the most relevant rows.

In [105]:

```

1 pd.DataFrame(grid_search.cv_results_)[
2     [
3         "mean_test_score",
4         "param_svc_gamma",
5         "param_svc_C",
6         "mean_fit_time",
7         "rank_test_score",
8     ]
9 ].set_index("rank_test_score").sort_index().T

```

Out[105]:

rank_test_score	1	2	3	4	5	6	7	8
mean_test_score	0.743961	0.732792	0.729091	0.720391	0.711715	0.704284	0.703034	0.697473
param_svc_gamma	0.1	0.01	0.1	0.01	0.1	0.001	0.1	0.01
param_svc_C	1.0	100	10	10	100	100	0.1	1.0
mean_fit_time	0.171719	0.274006	0.234079	0.177106	0.46566	0.194438	0.206641	0.175421

4 rows × 36 columns

- Other than searching for best hyperparameter values, `GridSearchCV` also fits a new model on the whole training set with the parameters that yielded the best results.
- So we can conveniently call `score` on the test set with a fitted `GridSearchCV` object.

```
In [106]: 1 grid_search.score(X_test, y_test)
```

Out[106]: 0.7376237623762376

Why `best_score_` and the score above are different?

`n_jobs=-1`

- Note the `n_jobs=-1` above.
- Hyperparameter optimization can be done *in parallel* for each of the configurations.
- This is very useful when scaling up to large numbers of machines in the cloud.

The `__` syntax

- Above: we have a nesting of transformers.
- We can access the parameters of the "inner" objects by using `__` to go "deeper":
- `svc__gamma`: the `gamma` of the `svc` of the pipeline
- `svc__C`: the `C` of the `svc` of the pipeline

```
In [ ]: 1 from sklearn.model_selection import GridSearchCV
2
3 pipe_svm = make_pipeline(StandardScaler(), SVC(**best_parameters))
4
5 param_grid = {
6     "svc__gamma": [0.001, 0.01, 0.1, 1.0, 10, 100],
7     "svc__C": [0.001, 0.01, 0.1, 1.0, 10, 100],
8 }
9
10 grid_search = GridSearchCV(
11     pipe_svm, param_grid, cv=5, n_jobs=-1, return_train_score=True
12 )
13
```

```
In [ ]: 1 grid_search.fit(X_train, y_train)
```

Visualizing the parameter grid as a heatmap

```
In [107]: 1 param_grid
```

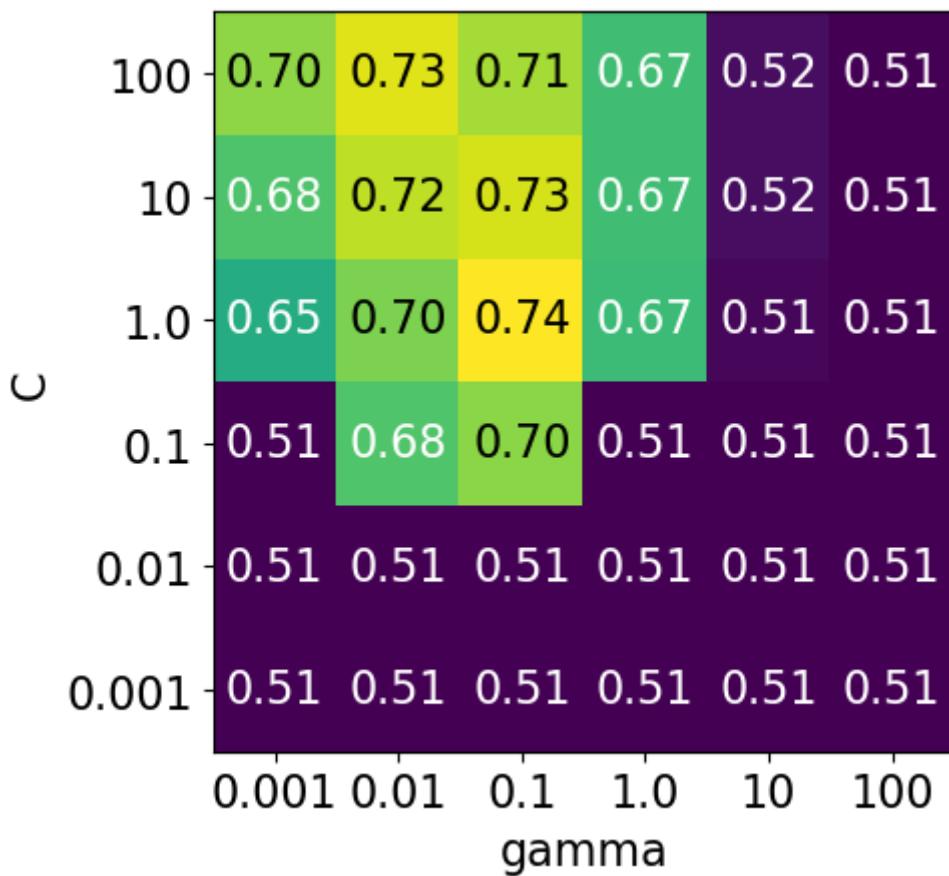
Out[107]: {'svc__gamma': [0.001, 0.01, 0.1, 1.0, 10, 100],
 'svc__C': [0.001, 0.01, 0.1, 1.0, 10, 100]}

In [108]:

```

1 results = pd.DataFrame(grid_search.cv_results_)
2
3 scores = np.array(results.mean_test_score).reshape(6, 6)
4
5 # plot the mean cross-validation scores
6 mglearn.tools.heatmap(
7     scores,
8     xlabel="gamma",
9     xticklabels=param_grid["svc__gamma"],
10    ylabel="C",
11    yticklabels=param_grid["svc__C"],
12    cmap="viridis",
13 );

```



- Each point in the heat map corresponds to one run of cross-validation, with a particular setting
- Colour encodes cross-validation accuracy.
 - Lighter colour means high accuracy
 - Darker colour means low accuracy
- SVC is quite sensitive to hyperparameter settings.
- Adjusting hyperparameters can change the accuracy from 0.51 to 0.74!

- Note that the range we pick for the parameters play an important role in hyperparameter optimization.
- For example, consider the following grid and the corresponding results.

In [109]:

```

1 def display_heatmap(param_grid, pipe, X_train, y_train):
2     grid_search = GridSearchCV(
3         pipe, param_grid, cv=5, n_jobs=-1, return_train_score=True
4     )
5     grid_search.fit(X_train, y_train)
6     results = pd.DataFrame(grid_search.cv_results_)
7     scores = np.array(results.mean_test_score).reshape(6, 6)
8
9     # plot the mean cross-validation scores
10    mglearn.tools.heatmap(
11        scores,
12        xlabel="gamma",
13        xticklabels=param_grid["svc__gamma"],
14        ylabel="C",
15        yticklabels=param_grid["svc__C"],
16        cmap="viridis",
17    );

```

Bad range for hyperparameters

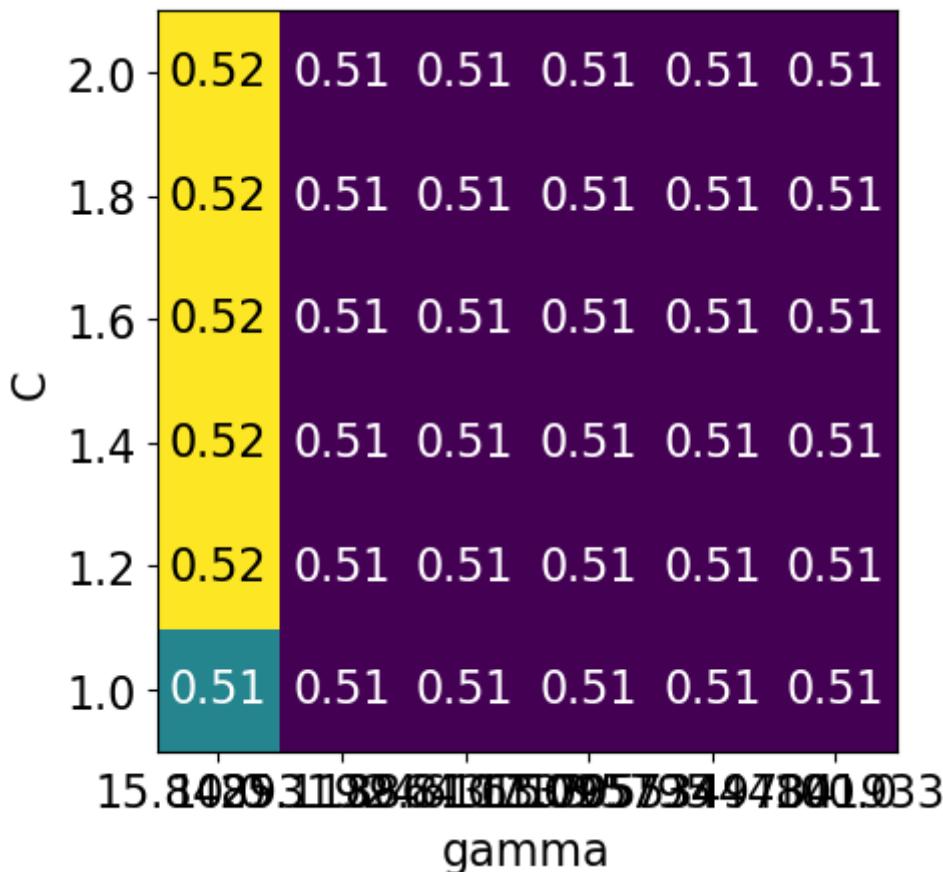
In [110]:

```

1 param_grid2 = {"svc__gamma": np.logspace(1, 2, 6), "svc__C": np.linspace(
2 display_heatmap(param_grid2, pipe_svm, X_train, y_train)
3 np.logspace(1, 2, 6)

```

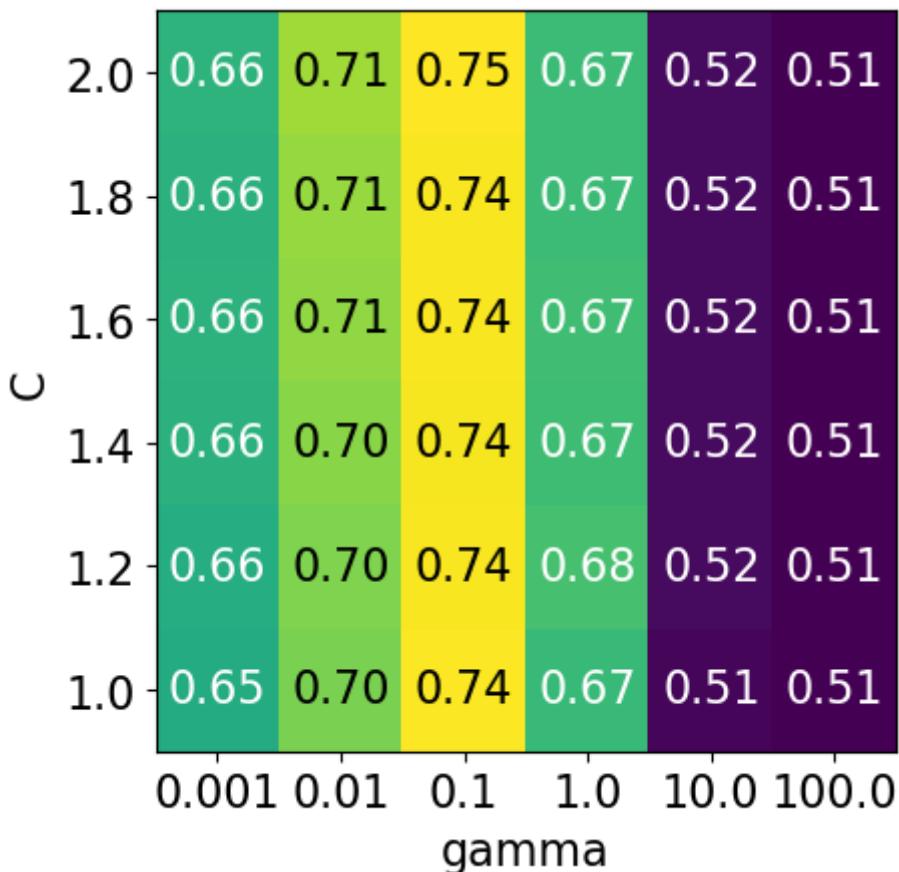
Out[110]: array([10. , 15.84893192, 25.11886432, 39.81071706,
 63.09573445, 100.])



Different range for hyperparameters yields better results!

In [111]:

```
1 param_grid3 = {"svc__gamma": np.logspace(-3, 2, 6), "svc__C": np.linspace(1.0, 2.0, 6)}
2
3 display_heatmap(param_grid3, pipe_svm, X_train, y_train)
```



It seems like we are getting even better cross-validation results with `C = 2.0` and `gamma = 0.1`

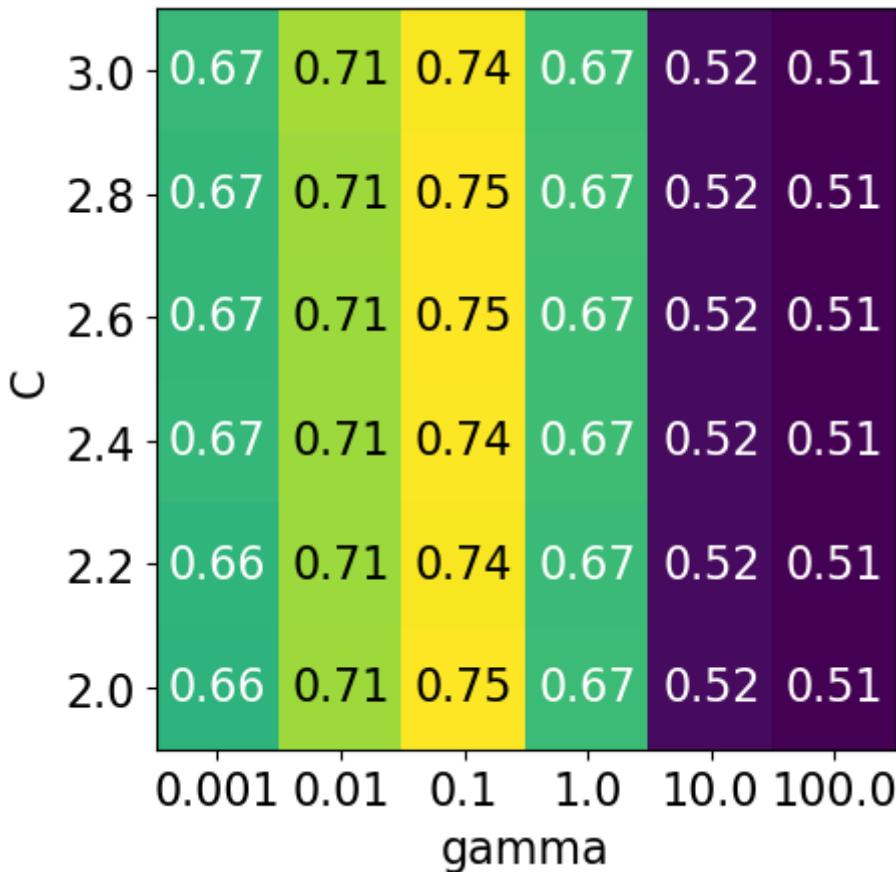
How about exploring different values of `C` close to 2.0?

In [112]:

```

1 param_grid4 = {"svc__gamma": np.logspace(-3, 2, 6), "svc__C": np.linspace(0.001, 100.0, 6)}
2
3 display_heatmap(param_grid4, pipe_svm, X_train, y_train)

```



That's good! We are finding some more options for `C` where the accuracy is 0.75. The tricky part is we do not know in advance what range of hyperparameters might work the best for the given problem, model, and the dataset.

True/False

- If you get optimal results at the edges of your parameter grid, it might be a good idea to adjust the range of values in your parameter grid.
- Grid search is guaranteed to find best hyperparameters values.

`GridSearchCV` allows the `param_grid` to be a list of dictionaries. Sometimes some hyperparameters are applicable only for certain models. For example, in the context of `svc`, `C` and `gamma` are applicable when the kernel is `rbf` whereas only `C` is applicable for `kernel="linear"`.

Problems with exhaustive grid search

- Required number of models to evaluate grows exponentially with the dimensionality of the configuration space.
- Example: Suppose you have
 - 5 hyperparameters
 - 10 different values for each hyperparameter
 - You'll be evaluating $10^5 = 100,000$ models! That is you'll be calling `cross_validate` 100,000 times!
- Exhaustive search may become infeasible fairly quickly.
- Other options?

Randomized hyperparameter search

- Randomized hyperparameter optimization
 - `sklearn.model_selection.RandomizedSearchCV` (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html)
- Samples configurations at random until certain budget (e.g., time) is exhausted

In [113]:

```

1 from sklearn.model_selection import RandomizedSearchCV
2
3
4 param_grid = {
5     "svc__gamma": [0.001, 0.01, 0.1, 1.0, 10, 100],
6     "svc__C": [0.001, 0.01, 0.1, 1.0, 10, 100],
7 }
8
9 print("Grid size: %d" % (np.prod(list(map(len, param_grid.values())))))
10 param_grid

```

Grid size: 36

Out[113]:

```
{'svc__gamma': [0.001, 0.01, 0.1, 1.0, 10, 100],
 'svc__C': [0.001, 0.01, 0.1, 1.0, 10, 100]}
```

In [114]:

```

1 random_search = RandomizedSearchCV(
2     pipe_svm, param_distributions=param_grid, n_jobs=-1, n_iter=10, cv=
3 )
4 random_search.fit(X_train, y_train);

```

```
In [115]: 1 pd.DataFrame(random_search.cv_results_)[
2     [
3         "mean_test_score",
4         "param_svc_gamma",
5         "param_svc_C",
6         "mean_fit_time",
7         "rank_test_score",
8     ]
9 ].set_index("rank_test_score").sort_index().T
```

Out[115]:

rank_test_score	1	2	3	4	5	6	6	6
mean_test_score	0.732792	0.711715	0.678851	0.652824	0.508371	0.50775	0.50775	0.50775
param_svc_gamma	0.01	0.1	0.01	0.001	100	0.001	0.1	100
param_svc_C	100	100	0.1	1.0	1.0	0.01	0.01	0.01
mean_fit_time	0.273256	0.480885	0.211935	0.17602	0.222688	0.215481	0.21545	0.183279

n_iter

- Note the `n_iter`, we didn't need this for `GridSearchCV`.
- Larger `n_iter` will take longer but it'll do more searching.
 - Remember you still need to multiply by number of folds!
- I have also set `random_state` but you don't have to do it.

Range of C

- Note the exponential range for `C`. This is quite common.
- There is no point trying $C = \{1, 2, 3 \dots, 100\}$ because $C = 1, 2, 3$ are too similar to each other.
- Often we're trying to find an order of magnitude, e.g. $C = \{0.01, 0.1, 1, 10, 100\}$.
- We can also write that as $C = \{10^{-2}, 10^{-1}, 10^0, 10^1, 10^2\}$.
- Or, in other words, C values to try are 10^n for $n = -2, -1, 0, 1, 2$ which is basically what we have above.

(Optional) Another thing we can do is give probability distributions to draw from:

```
In [116]: 1 from scipy.stats import expon, lognorm, loguniform, randint, uniform
```

```
In [117]: 1 param_dist = {
2     "svc_C": uniform(0.1, 1e4), # loguniform(1e-3, 1e3),
3     "svc_gamma": loguniform(1e-5, 1e3),
4 }
```

```
In [118]: 1 random_search = RandomizedSearchCV(
2     pipe_svm, param_dist, n_iter=100, verbose=1, n_jobs=-1, random_state=123)
```

```
In [119]: 1 random_search.fit(X_train, y_train)
```

Fitting 5 folds for each of 100 candidates, totalling 500 fits

```
Out[119]: RandomizedSearchCV(estimator=Pipeline(steps=[('standardscaler',
    StandardScaler()),
    ('svc', SVC())]),
    n_iter=100, n_jobs=-1,
    param_distributions={'svc__C': <scipy.stats._distn_infrastructure.rv_continuous_frozen object at 0x18dcbf8e0>,
    'svc__gamma': <scipy.stats._distn_infrastructure.rv_continuous_frozen object at 0x18dcdbc90>},
    random_state=123, verbose=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [120]: 1 random_search.best_score_
```

```
Out[120]: 0.7383804780493433
```

```
In [121]: 1 pd.DataFrame(random_search.cv_results_)[
2     [
3         "mean_test_score",
4         "param_svc__gamma",
5         "param_svc__C",
6         "mean_fit_time",
7         "rank_test_score",
8     ]
9 ].set_index("rank_test_score").sort_index().T
```

	rank_test_score	1	2	3	4	5	6
mean_test_score	0.73838	0.7359	0.735277	0.733415	0.731556	0.729716	
param_svc__gamma	0.00271	0.001946	0.00283	0.003148	0.003834	0.015524	
param_svc__C	3427.738338	6964.791856	2865.466167	4258.402903	7224.533826	1511.374523	
mean_fit_time	1.10842	1.367456	1.022985	1.490474	2.501841	1.809782	

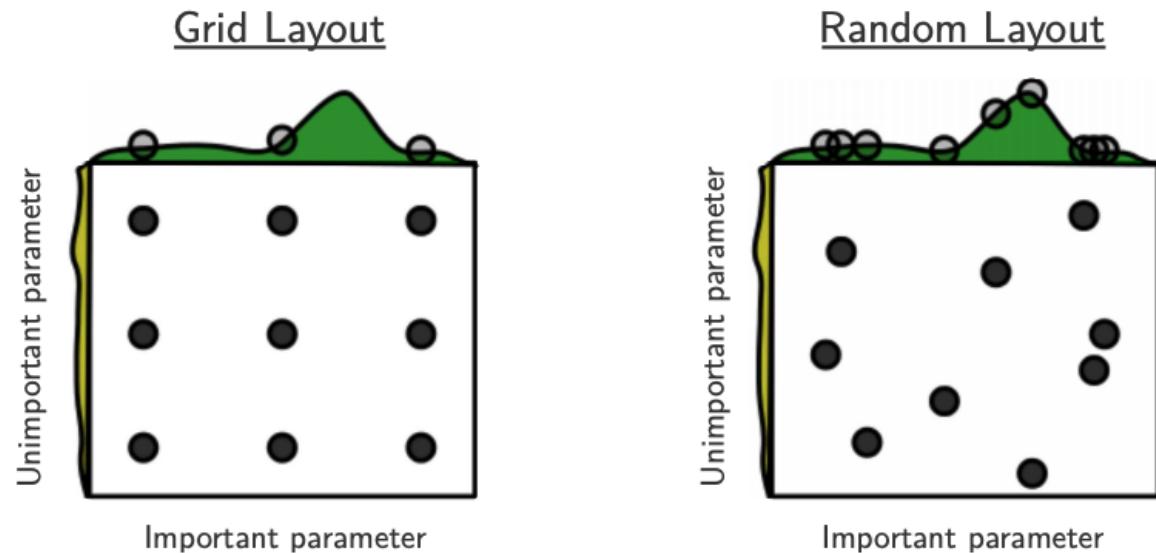
4 rows × 100 columns

- This is a bit fancy. What's nice is that you can have it concentrate more on certain values by setting the distribution.

Advantages of RandomizedSearchCV

- Faster compared to GridSearchCV .
- Adding parameters that do not influence the performance does not affect efficiency.
- Works better when some parameters are more important than others.
- In general, I recommend using RandomizedSearchCV rather than GridSearchCV .

Advantages of RandomizedSearchCV



Source: [Bergstra and Bengio, Random Search for Hyper-Parameter Optimization, JMLR 2012 \(<http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf>\).](http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf)

- The yellow on the left shows how your scores are going to change when you vary the unimportant hyperparameter.
- The green on the top shows how your scores are going to change when you vary the important hyperparameter.
- You don't know in advance which hyperparameters are important for your problem.
- In the left figure, 6 of the 9 searches are useless because they are only varying the unimportant parameter.
- In the right figure, all 9 searches are useful.

Fancier methods (optional)

- Both GridSearchCV and RandomizedSearchCV do each trial independently.
- What if you could learn from your experience, e.g. learn that `max_depth=3` is bad?
 - That could save time because you wouldn't try combinations involving `max_depth=3` in the future.
- We can do this with `scikit-optimize`, which is a completely different package from `scikit-learn`

- It uses a technique called "model-based optimization" and we'll specifically use "Bayesian optimization".
 - In short, it uses machine learning to predict what hyperparameters will be good.
 - Machine learning on machine learning!
- This is an active research area of research, and there are sophisticated packages for this.

Here are some examples

- [hyperopt-sklearn \(<https://github.com/hyperopt/hyperopt-sklearn>\)](https://github.com/hyperopt/hyperopt-sklearn)
- [auto-sklearn \(<https://github.com/automl/auto-sklearn>\)](https://github.com/automl/auto-sklearn)
- [SigOptSearchCV \(\[https://sigopt.com/docs/overview/scikit_learn\]\(https://sigopt.com/docs/overview/scikit_learn\)\)](https://sigopt.com/docs/overview/scikit_learn)
- [TPOT \(<https://github.com/rhiever/tpot>\)](https://github.com/rhiever/tpot)
- [hyperopt \(<https://github.com/hyperopt/hyperopt>\)](https://github.com/hyperopt/hyperopt)
- [hyperband \(<https://github.com/zygmuntz/hyperband>\)](https://github.com/zygmuntz/hyperband)
- [SMAC \(<http://www.cs.ubc.ca/labs/beta/Projects/SMAC/>\)](http://www.cs.ubc.ca/labs/beta/Projects/SMAC/)
- [MOE \(<https://github.com/Yelp/MOE>\)](https://github.com/Yelp/MOE)
- [pybo \(<https://github.com/mwhoffman/pybo>\)](https://github.com/mwhoffman/pybo)
- [spearmint \(<https://github.com/HIPS/Spearmint>\)](https://github.com/HIPS/Spearmint)
- [BayesOpt \(<https://github.com/rmcantin/bayesopt>\)](https://github.com/rmcantin/bayesopt)

In []:

1

Questions for class discussion (hyperparameter optimization)

- Suppose you have 10 hyperparameters, each with 4 possible values. If you run `GridSearchCV` with this parameter grid, how many cross-validation experiments would it carry out?
- `GridSearchCV` exhaustively searches the grid and so it's guaranteed to give you the optimal hyperparameters for the given problem. True or false?
- Is it possible to get different hyperparameters in different runs of `RandomizedSearchCV`?
- Suppose you have 10 hyperparameters and each takes 4 values. If you run `RandomizedSearchCV` with this parameter grid, how many cross-validation experiments it would carry out?

Optimization bias/Overfitting of the validation set

Overfitting of the validation error

- Why do we need to evaluate the model on the test set in the end?

- Why not just use cross-validation on the whole dataset?
- While carrying out hyperparameter optimization, we usually try over many possibilities.
- If our dataset is small and if your validation set is hit too many times, we suffer from **optimization bias** or **overfitting the validation set**.

Optimization bias of parameter learning

- Overfitting of the training error
- An example:
 - During training, we could search over tons of different decision trees.
 - So we can get "lucky" and find one with low training error by chance.

Optimization bias of hyper-parameter learning

- Overfitting of the validation error
- An example:
 - Here, we might optimize the validation error over 1000 values of `max_depth`.
 - One of the 1000 trees might have low validation error by chance.

Example 1: Optimization bias (optional)

Consider a multiple-choice (a,b,c,d) "test" with 10 questions:

- If you choose answers randomly, expected grade is 25% (no bias).
- If you fill out two tests randomly and pick the best, expected grade is 33%.
 - Optimization bias of ~8%.
- If you take the best among 10 random tests, expected grade is ~47%.
- If you take the best among 100, expected grade is ~62%.
- If you take the best among 1000, expected grade is ~73%.
- If you take the best among 10000, expected grade is ~82%.
 - You have so many "chances" that you expect to do well.

But on new questions the "random choice" accuracy is still 25%.

In [132]:

```

1 # (optional) Code attribution: Rodolfo Lourenzutti
2 number_tests = [1, 2, 10, 100, 1000, 10000]
3 for ntests in number_tests:
4     y = np.zeros(10000)
5     for i in range(10000):
6         y[i] = np.max(np.random.binomial(10.0, 0.25, ntests))
7     print(
8         "The expected grade among the best of %d tests is : %0.2f"
9         % (ntests, np.mean(y) / 10.0)
10    )

```

The expected grade among the best of 1 tests is : 0.25
 The expected grade among the best of 2 tests is : 0.33
 The expected grade among the best of 10 tests is : 0.47
 The expected grade among the best of 100 tests is : 0.62
 The expected grade among the best of 1000 tests is : 0.73
 The expected grade among the best of 10000 tests is : 0.83

Example 2: Optimization bias (optional)

- If we instead used a 100-question test then:
 - Expected grade from best over 1 randomly-filled test is 25%.
 - Expected grade from best over 2 randomly-filled test is ~27%.
 - Expected grade from best over 10 randomly-filled test is ~32%.
 - Expected grade from best over 100 randomly-filled test is ~36%.
 - Expected grade from best over 1000 randomly-filled test is ~40%.
 - Expected grade from best over 10000 randomly-filled test is ~43%.
- The optimization bias **grows with the number of things we try**.
 - “Complexity” of the set of models we search over.
- But, optimization bias **shrinks quickly with the number of examples**.
 - But it's still non-zero and growing if you over-use your validation set!

In []:

```

1 # (optional) Code attribution: Rodolfo Lourenzutti
2 number_tests = [1, 2, 10, 100, 1000, 10000]
3 for ntests in number_tests:
4     y = np.zeros(10000)
5     for i in range(10000):
6         y[i] = np.max(np.random.binomial(100.0, 0.25, ntests))
7     print(
8         "The expected grade among the best of %d tests is : %0.2f"
9         % (ntests, np.mean(y) / 100.0)
10    )

```

Optimization bias on the Spotify dataset

```
In [122]: 1 X_train_tiny, X_test_big, y_train_tiny, y_test_big = train_test_split(
2     X_spotify, y_spotify, test_size=0.99, random_state=42
3 )
```

```
In [123]: 1 X_train_tiny.shape
```

Out[123]: (20, 13)

```
In [125]: 1 X_train_tiny.head()
```

	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	m
130	0.055100	0.547	251093	0.643	0.000000	1	0.2670	-8.904	
1687	0.000353	0.420	210240	0.929	0.000747	7	0.1220	-3.899	
871	0.314000	0.430	193427	0.734	0.000286	9	0.0808	-10.043	
1123	0.082100	0.725	246653	0.711	0.000000	10	0.0931	-4.544	
1396	0.286000	0.616	236960	0.387	0.000000	9	0.2770	-6.079	

```
In [126]: 1 pipe = make_pipeline(StandardScaler(), SVC())
```

```
In [127]: 1 from sklearn.model_selection import RandomizedSearchCV
2
3 param_grid = {
4     "svc__gamma": 10.0 ** np.arange(-20, 10),
5     "svc__C": 10.0 ** np.arange(-20, 10),
6 }
7 print("Grid size: %d" % (np.prod(list(map(len, param_grid.values())))))
8 param_grid
```

Grid size: 900

Out[127]: {'svc__gamma': array([1.e-20, 1.e-19, 1.e-18, 1.e-17, 1.e-16, 1.e-15, 1.e-14, 1.e-13, 1.e-12, 1.e-11, 1.e-10, 1.e-09, 1.e-08, 1.e-07, 1.e-06, 1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03, 1.e+04, 1.e+05, 1.e+06, 1.e+07, 1.e+08, 1.e+09]), 'svc__C': array([1.e-20, 1.e-19, 1.e-18, 1.e-17, 1.e-16, 1.e-15, 1.e-14, 1.e-13, 1.e-12, 1.e-11, 1.e-10, 1.e-09, 1.e-08, 1.e-07, 1.e-06, 1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03, 1.e+04, 1.e+05, 1.e+06, 1.e+07, 1.e+08, 1.e+09])}

```
In [128]: 1 random_search = RandomizedSearchCV(
2     pipe, param_distributions=param_grid, n_jobs=-1, n_iter=900, cv=5,
3 )
4 random_search.fit(X_train_tiny, y_train_tiny);
```

In [129]:

```

1 pd.DataFrame(random_search.cv_results_)[
2     [
3         "mean_test_score",
4         "param_svc_gamma",
5         "param_svc_C",
6         "mean_fit_time",
7         "rank_test_score",
8     ]
9 ].set_index("rank_test_score").sort_index().T

```

Out[129]:

rank_test_score	1	1	3	3	3	3
mean_test_score	0.8	0.8	0.75	0.75	0.75	0.75
param_svc_gamma	0.0	0.0	0.001	0.001	0.001	0.0
param_svc_C	10000000000.0	1000000000.0	10000000000.0	10000.0	1000000.0	10000000.0
mean_fit_time	0.00842	0.008212	0.007549	0.011502	0.007781	0.009093

4 rows × 900 columns

Given the results: one might claim that we found a model that performs with 0.8 accuracy on our dataset.

- Do we really believe that 0.80 is a good estimate of our test data?
- Do we really believe that `gamma = 0.0` and `C=1_000_000_000` are the best hyperparameters?
- Let's find out the test score with this best model.

In [130]:

```
1 random_search.score(X_test, y_test)
```

Out[130]:

0.6163366336633663

- The results above are overly optimistic.
 - because in each fold our training data is very small, and our validation data even smaller,
 - and the fact that we try 900 times with different complexities of models,
 - it is possible to get lucky on the validation folds!
- As we suspected, the best cross-validation score is not a good estimate of our test data; it is overly optimistic.
- We can trust this test score because the test set is of good size.

In [131]:

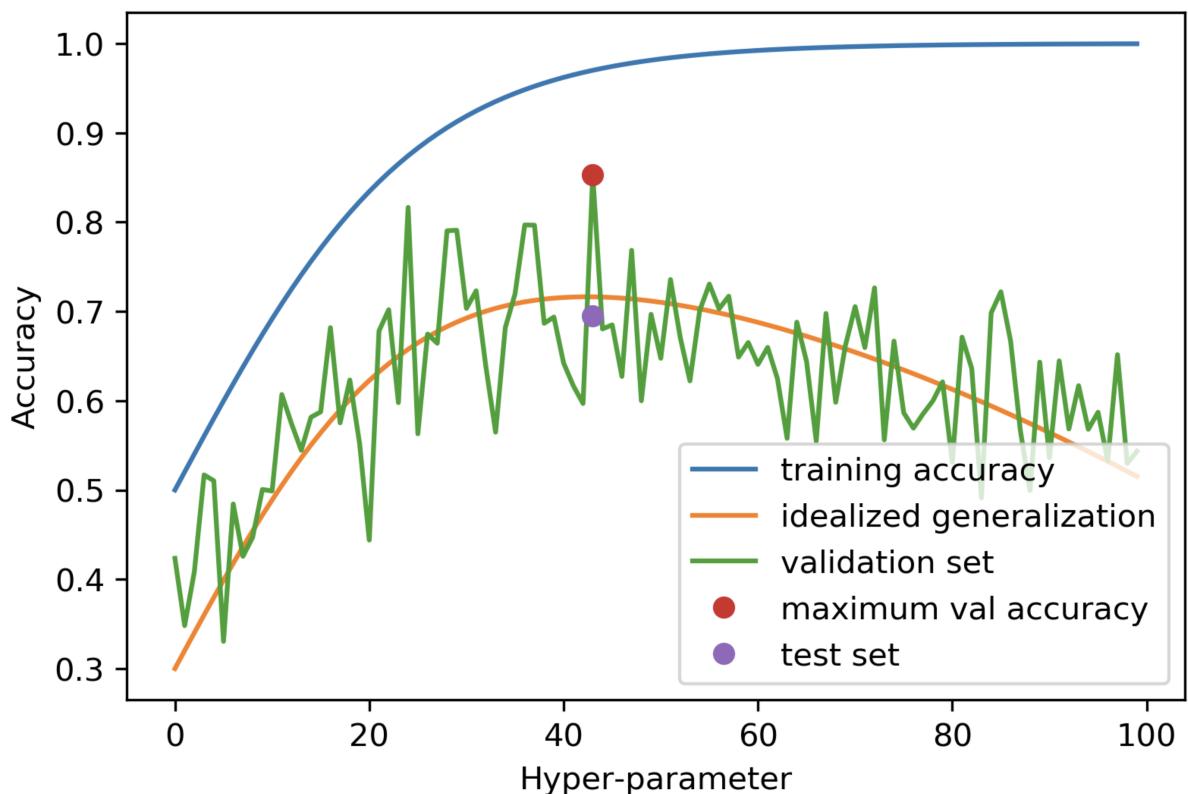
```
1 X_test_big.shape
```

Out[131]:

(1997, 13)

Overfitting of the validation data

The following plot demonstrates what happens during overfitting of the validation data.



Source (<https://amueller.github.io/COMS4995-s20/slides/aml-03-supervised-learning/#20>).

- Thus, not only can we not trust the cv scores, we also cannot trust cv's ability to choose of the best hyperparameters.

Why do we need a test set?

- This is why we need a test set.
- The frustrating part is that if our dataset is small, so is our test set 😞.
- Unfortunately, we don't have much better alternatives when we have a small dataset.

When test score is much lower than CV score

- What to do if your test score is much lower than your cross-validation score:
 - Try simpler models and use the test set a couple of times; it's not the end of the world.
 - Communicate this clearly when you report the results.

Large datasets solve many of these problems

- With infinite amounts of training data, overfitting would not be a problem and you could have your test score = your train score.
 - Overfitting happens because you only see a bit of data and you learn patterns that are overly specific to your sample.
 - If you saw "all" the data, then the notion of "overly specific" would not apply.
- So, more data will make your test score better and more robust.

? ? Questions for you

Would you trust the model?

- You have a dataset and you give me half of it. I build a model using all the data you have given me and I tell you that the model accuracy is 0.99. Would it classify the rest of the data with similar accuracy?
 1. Probably
 2. Probably not

Would you trust the model?

- You have a dataset and you give me half of it. I build a model using 80% of the data given to me and report the accuracy of 0.95 on the remaining 20% of the data. Would it classify the rest of the data with similar accuracy?
 1. Probably
 2. Probably not

Would you trust the model?

- You have a dataset and you give me 1/10th of it. The dataset given to me is rather small and so I split it into 96% train and 4% validation split. I carry out hyperparameter optimization using a single 4% validation split and report validation accuracy of 0.97. Would it classify the rest of the data with similar accuracy?
 1. Probably
 2. Probably not

Final comments and summary

Automated hyperparameter optimization

- Advantages
 - reduce human effort
 - less prone to error and improve reproducibility
 - data-driven approaches may be effective
- Disadvantages
 - may be hard to incorporate intuition
 - be careful about overfitting on the validation set

Often, especially on typical datasets, we get back `scikit-learn`'s default hyperparameter values. This means that the defaults are well chosen by `scikit-learn` developers!

- The problem of finding the best values for the important hyperparameters is tricky because
 - You may have a lot of them (e.g. deep learning).
 - You may have multiple hyperparameters which may interact with each other in unexpected ways.
- The best settings depend on the specific data/problem.

Optional readings and resources

- [Preventing "overfitting" of cross-validation data](http://www.robotics.stanford.edu/~ang/papers/cv-final.pdf)
(<http://www.robotics.stanford.edu/~ang/papers/cv-final.pdf>) by Andrew Ng

CPSC 330

Applied Machine Learning

Lecture 9: Classification Metrics

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

```
In [99]: 1 import os
2 import sys
3
4 sys.path.append("../code/.")
5
6 import IPython
7 import matplotlib.pyplot as plt
8 import mglearn
9 import numpy as np
10 import pandas as pd
11 from IPython.display import HTML, display
12 from plotting_functions import *
13 from sklearn.dummy import DummyClassifier
14 from sklearn.linear_model import LogisticRegression
15 from sklearn.model_selection import cross_val_score, cross_validate, train_test_split
16 from sklearn.pipeline import Pipeline, make_pipeline
17 from sklearn.preprocessing import StandardScaler
18 from utils import *
19
20 %matplotlib inline
21 pd.set_option("display.max_colwidth", 200)
22
23 from IPython.display import Image
```

```
In [100]: 1 # Changing global matplotlib settings for confusion matrix.
2 plt.rcParams["xtick.labelsize"] = 18
3 plt.rcParams["ytick.labelsize"] = 18
```

```
In [101]: 1 import warnings
2
3 warnings.simplefilter(action="ignore", category=FutureWarning)
```

Learning outcomes

From this lecture, students are expected to be able to:

- Explain why accuracy is not always the best metric in ML.
- Explain components of a confusion matrix.
- Define precision, recall, and f1-score and use them to evaluate different classifiers.
- Broadly explain macro-average, weighted average.
- Interpret and use precision-recall curves.
- Explain average precision score.
- Interpret and use ROC curves and ROC AUC using `scikit-learn`.
- Identify whether there is class imbalance and whether you need to deal with it.
- Explain and use `class_weight` to deal with data imbalance.

Evaluation metrics for binary classification: Motivation

Dataset for demonstration

- Let's classify fraudulent and non-fraudulent transactions using Kaggle's [Credit Card Fraud Detection](https://www.kaggle.com/mlg-ulb/creditcardfraud) (<https://www.kaggle.com/mlg-ulb/creditcardfraud>) data set.

```
In [102]: 1 cc_df = pd.read_csv("../data/creditcard.csv", encoding="latin-1")
2 train_df, test_df = train_test_split(cc_df, test_size=0.3, random_state
3 train_df.head()
```

```
Out[102]:
```

	Time	V1	V2	V3	V4	V5	V6	V7	
64454	51150.0	-3.538816	3.481893	-1.827130	-0.573050	2.644106	-0.340988	2.102135	-2.9391
37906	39163.0	-0.363913	0.853399	1.648195	1.118934	0.100882	0.423852	0.472790	-0.9721
79378	57994.0	1.193021	-0.136714	0.622612	0.780864	-0.823511	-0.706444	-0.206073	-0.0161
245686	152859.0	1.604032	-0.808208	-1.594982	0.200475	0.502985	0.832370	-0.034071	0.2341
60943	49575.0	-2.669614	-2.734385	0.662450	-0.059077	3.346850	-2.549682	-1.430571	-0.1181

5 rows × 31 columns

```
In [103]: 1 train_df.shape
```

```
Out[103]: (199364, 31)
```

- Good sized dataset

- For confidentiality reasons, it only provides features transformed with PCA, which is a popular dimensionality reduction technique.

EDA

In [104]: 1 train_df.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 199364 entries, 64454 to 129900
Data columns (total 31 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   Time      199364 non-null   float64
 1   V1        199364 non-null   float64
 2   V2        199364 non-null   float64
 3   V3        199364 non-null   float64
 4   V4        199364 non-null   float64
 5   V5        199364 non-null   float64
 6   V6        199364 non-null   float64
 7   V7        199364 non-null   float64
 8   V8        199364 non-null   float64
 9   V9        199364 non-null   float64
 10  V10       199364 non-null   float64
 11  V11       199364 non-null   float64
 12  V12       199364 non-null   float64
 13  V13       199364 non-null   float64
 14  V14       199364 non-null   float64
 15  V15       199364 non-null   float64
 16  V16       199364 non-null   float64
 17  V17       199364 non-null   float64
 18  V18       199364 non-null   float64
 19  V19       199364 non-null   float64
 20  V20       199364 non-null   float64
 21  V21       199364 non-null   float64
 22  V22       199364 non-null   float64
 23  V23       199364 non-null   float64
 24  V24       199364 non-null   float64
 25  V25       199364 non-null   float64
 26  V26       199364 non-null   float64
 27  V27       199364 non-null   float64
 28  V28       199364 non-null   float64
 29  Amount    199364 non-null   float64
 30  Class     199364 non-null   int64
dtypes: float64(30), int64(1)
memory usage: 48.7 MB
```

In [105]: 1 train_df.describe(include="all")

Out[105]:

	Time	V1	V2	V3	V4	V5
count	199364.000000	199364.000000	199364.000000	199364.000000	199364.000000	199364.000000
mean	94888.815669	0.000492	-0.000726	0.000927	0.000630	0.000036
std	47491.435489	1.959870	1.645519	1.505335	1.413958	1.361718
min	0.000000	-56.407510	-72.715728	-31.813586	-5.683171	-42.147898
25%	54240.000000	-0.918124	-0.600193	-0.892476	-0.847178	-0.691241
50%	84772.500000	0.018854	0.065463	0.179080	-0.019531	-0.056703
75%	139349.250000	1.315630	0.803617	1.028023	0.744201	0.610407
max	172792.000000	2.451888	22.057729	9.382558	16.491217	34.801666

8 rows × 31 columns

- We do not have categorical features. All features are numeric.
- We have to be careful about the `Time` and `Amount` features.
- We could scale `Amount`.
- Do we want to scale time?
 - We will in this lecture, though it's probably not the best thing to do.
 - We'll learn about time series briefly later in the course.

Let's separate `x` and `y` for train and test splits.

In [106]: 1 X_train_big, y_train_big = train_df.drop(columns=["Class"]), train_df["
2 X_test, y_test = test_df.drop(columns=["Class"]), test_df["Class"]]

- It's easier to demonstrate evaluation metrics using an explicit validation set instead of using cross-validation.
- So let's create a validation set.
- Our data is large enough so it shouldn't be a problem.

In [107]: 1 X_train, X_valid, y_train, y_valid = train_test_split(
2 X_train_big, y_train_big, test_size=0.3, random_state=123
3)

Baseline

```
In [108]: 1 dummy = DummyClassifier()
2 pd.DataFrame(cross_validate(dummy, X_train, y_train, return_train_score=True))
```

```
Out[108]: fit_time      0.019434
score_time     0.004295
test_score     0.998302
train_score    0.998302
dtype: float64
```

Observations

- DummyClassifier is getting 0.998 cross-validation accuracy!!
- Should we be happy with this accuracy and deploy this DummyClassifier model for fraud detection?

What's the class distribution?

```
In [109]: 1 train_df["Class"].value_counts(normalize=True)
```

```
Out[109]: 0    0.9983
1    0.0017
Name: Class, dtype: float64
```

- We have class imbalance.
- We have MANY non-fraud transactions and only a handful of fraud transactions.
- So in the training set, most_frequent strategy is labeling 199,025 (99.83%) instances correctly and only 339 (0.17%) instances incorrectly.
- Is this what we want?
- The "fraud" class is the important class that we want to spot.

Let's scale the features and try LogisticRegression .

```
In [110]: 1 pipe = make_pipeline(StandardScaler(), LogisticRegression())
2 pd.DataFrame(cross_validate(pipe, X_train, y_train, return_train_score=True))
```

```
Out[110]: fit_time      0.882833
score_time     0.015415
test_score     0.999176
train_score    0.999249
dtype: float64
```

- We are getting a slightly better score with logistic regression.
- What score should be considered an acceptable score here?
- Are we actually spotting many "fraud" transactions?

- `.score` by default returns accuracy which is
$$\frac{\text{correct predictions}}{\text{total examples}}$$
- Is accuracy a good metric here?
- Is there anything more informative than accuracy that we can use here?

Let's dig a little deeper.

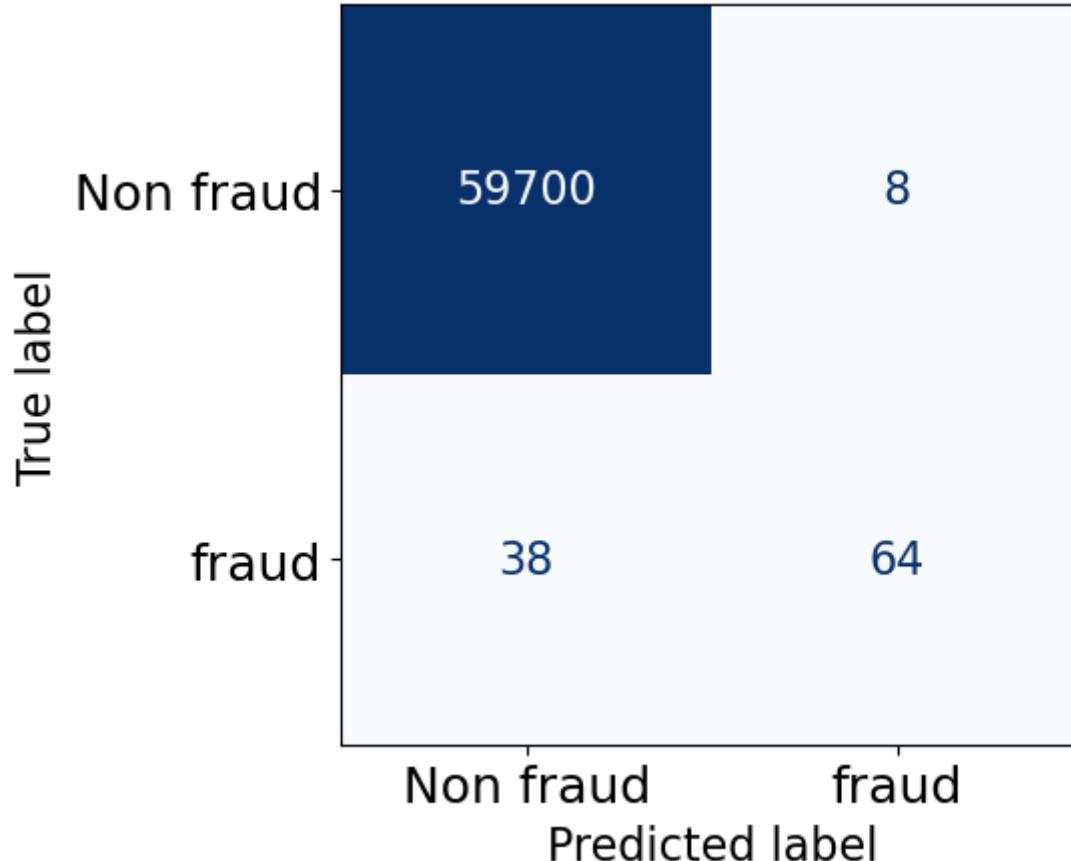
Confusion matrix

One way to get a better understanding of the errors is by looking at

- false positives (type I errors), where the model incorrectly spots examples as fraud
- false negatives (type II errors), where it's missing to spot fraud examples

In [111]:

```
1 from sklearn.metrics import plot_confusion_matrix
2
3 pipe.fit(X_train, y_train)
4 disp = plot_confusion_matrix(
5     pipe,
6     X_valid,
7     y_valid,
8     display_labels=["Non fraud", "fraud"],
9     values_format="d",
10    cmap=plt.cm.Blues,
11    colorbar=False,
12 );
```

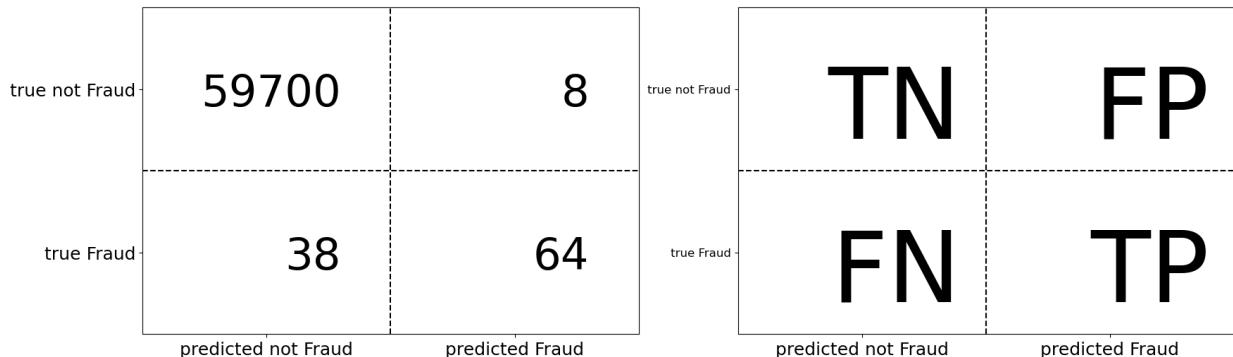


In [112]:

```

1 from sklearn.metrics import confusion_matrix
2
3 predictions = pipe.predict(X_valid)
4 TN, FP, FN, TP = confusion_matrix(y_valid, predictions).ravel()
5 plot_confusion_matrix_example(TN, FP, FN, TP)

```



- Perfect prediction has all values down the diagonal
- Off diagonal entries can often tell us about what is being mis-predicted

What is "positive" and "negative"?

- Two kinds of binary classification problems
 - Distinguishing between two classes
 - Spotting a class (spot fraud transaction, spot spam, spot disease)
- In case of spotting problems, the thing that we are interested in spotting is considered "positive".
- Above we wanted to spot fraudulent transactions and so they are "positive".

You can get a numpy array of confusion matrix as follows:

In [113]:

```

1 from sklearn.metrics import confusion_matrix
2
3 predictions = pipe.predict(X_valid)
4 TN, FP, FN, TP = confusion_matrix(y_valid, predictions).ravel()
5 print("Confusion matrix for fraud data set")
6 print(confusion_matrix)

```

Confusion matrix for fraud data set

```

[[59700     8]
 [   38    64]]

```

Confusion matrix with cross-validation

- You can also calculate confusion matrix with cross-validation using the `cross_val_predict` method.
- But then you cannot conveniently use `plot_confusion_matrix`.

In [114]:

```

1 from sklearn.model_selection import cross_val_predict
2
3 confusion_matrix(y_train, cross_val_predict(pipe, X_train, y_train))

```

Out[114]: array([[139296, 21],
 [94, 143]])

Precision, recall, f1 score

- We have been using `.score` to assess our models, which returns accuracy by default.
- Accuracy is misleading when we have class imbalance.
- We need other metrics to assess our models.

- We'll discuss three commonly used metrics which are based on confusion matrix:
 - recall
 - precision
 - f1 score
- Note that these metrics will only help us assessing our model.
- Later we'll talk about a few ways to address class imbalance problem.

In [115]:

```

1 from sklearn.metrics import confusion_matrix
2
3 pipe_lr = make_pipeline(StandardScaler(), LogisticRegression())
4 pipe_lr.fit(X_train, y_train)
5 predictions = pipe_lr.predict(X_valid)
6 TN, FP, FN, TP = confusion_matrix(y_valid, predictions).ravel()
7 print(disp.confusion_matrix)

```

[[59700 8]
 [38 64]]

Recall

Among all positive examples, how many did you identify?

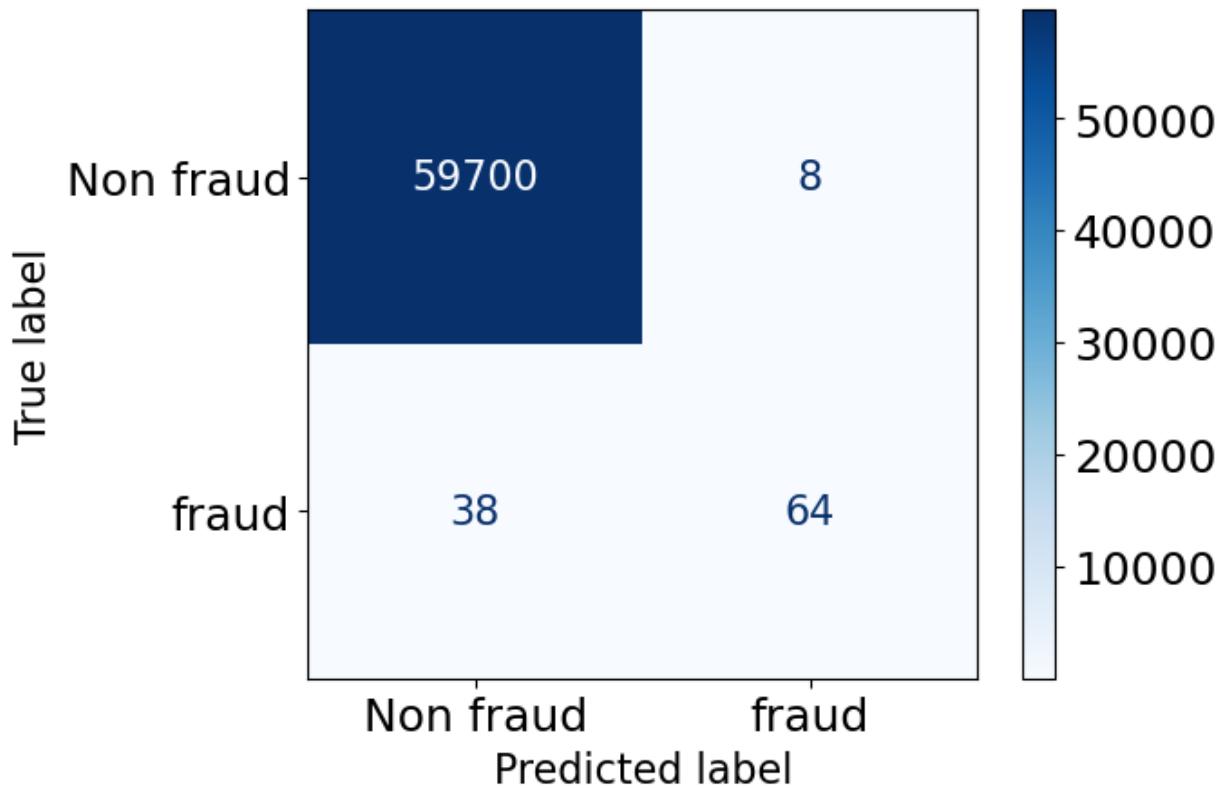
$$\text{recall} = \frac{TP}{TP + FN} = \frac{TP}{\#positives}$$

In [116]:

```

1 plot_confusion_matrix(
2     pipe,
3     X_valid,
4     y_valid,
5     display_labels=["Non fraud", "fraud"],
6     values_format="d",
7     cmap=plt.cm.Blues,
8 );

```



In [117]:

```

1 print("TP = %0.4f, FN = %0.4f" % (TP, FN))
2 recall = TP / (TP + FN)
3 print("Recall: %0.4f" % (recall))

```

TP = 64.0000, FN = 38.0000

Recall: 0.6275

Precision

Among the positive examples you identified, how many were actually positive?

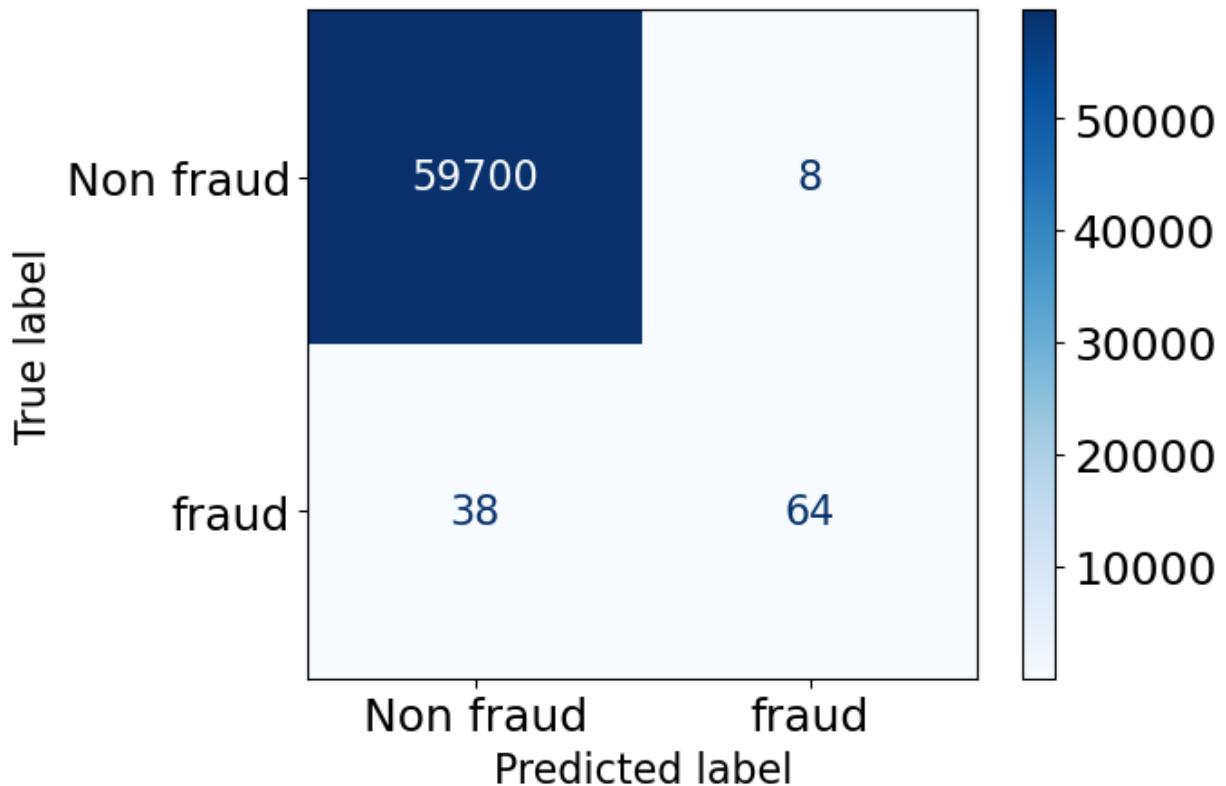
$$precision = \frac{TP}{TP + FP}$$

In [118]:

```

1 plot_confusion_matrix(
2     pipe,
3     X_valid,
4     y_valid,
5     display_labels=["Non fraud", "fraud"],
6     values_format="d",
7     cmap=plt.cm.Blues,
8 );

```



In [119]:

```

1 print("TP = %0.4f, FP = %0.4f" % (TP, FP))
2 precision = TP / (TP + FP)
3 print("Precision: %0.4f" % (precision))

```

TP = 64.0000, FP = 8.0000

Precision: 0.8889

F1-score

- F1-score combines precision and recall to give one score, which could be used in hyperparameter optimization, for instance.
- F1-score is a harmonic mean of precision and recall.

$$f1 = 2 \times \frac{precision \times recall}{precision + recall}$$

In [120]:

```

1 print("precision: %0.4f" % (precision))
2 print("recall: %0.4f" % (recall))
3 f1_score = (2 * precision * recall) / (precision + recall)
4 print("f1: %0.4f" % (f1_score))

```

```

precision: 0.8889
recall: 0.6275
f1: 0.7356

```

Let's look at all metrics at once on our dataset.

In [121]:

```

1 ## Calculate evaluation metrics by ourselves
2 data = {
3     "calculation": [],
4     "accuracy": [],
5     "error": [],
6     "precision": [],
7     "recall": [],
8     "f1 score": []
9 }
10 data["calculation"].append("manual")
11 data["accuracy"].append((TP + TN) / (TN + FP + FN + TP))
12 data["error"].append((FP + FN) / (TN + FP + FN + TP))
13 data["precision"].append(precision) # TP / (TP + FP)
14 data["recall"].append(recall) # TP / (TP + FN)
15 data["f1 score"].append(f1_score) # (2 * precision * recall) / (precision + recall)
16 df = pd.DataFrame(data)
17 df

```

Out[121]:

	calculation	accuracy	error	precision	recall	f1 score
0	manual	0.999231	0.000769	0.888889	0.627451	0.735632

- scikit-learn has functions for these metrics (<https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics>).

In [122]:

```

1 from sklearn.metrics import accuracy_score, f1_score, precision_score,
2
3 data[ "accuracy" ].append(accuracy_score(y_valid, pipe_lr.predict(X_valid)))
4 data[ "error" ].append(1 - accuracy_score(y_valid, pipe_lr.predict(X_valid)))
5 data[ "precision" ].append(
6     precision_score(y_valid, pipe_lr.predict(X_valid), zero_division=1)
7 )
8 data[ "recall" ].append(recall_score(y_valid, pipe_lr.predict(X_valid)))
9 data[ "f1 score" ].append(f1_score(y_valid, pipe_lr.predict(X_valid)))
10 data[ "calculation" ].append("sklearn")
11 df = pd.DataFrame(data)
12 df.set_index(["calculation"])

```

Out[122]:

	accuracy	error	precision	recall	f1 score
--	----------	-------	-----------	--------	----------

calculation					
--------------------	--	--	--	--	--

manual	0.999231	0.000769	0.888889	0.627451	0.735632
sklearn	0.999231	0.000769	0.888889	0.627451	0.735632

The scores match.

Classification report

- There is a convenient function called `classification_report` in `sklearn` which gives this info.

In [123]:

```
1 pipe_lr.classes_
```

Out[123]:

```
array([0, 1])
```

In [124]:

```

1 from sklearn.metrics import classification_report
2
3 print(
4     classification_report(
5         y_valid, pipe_lr.predict(X_valid), target_names=[ "non-fraud" , "fraud" ]
6     )
7 )

```

	precision	recall	f1-score	support
non-fraud	1.00	1.00	1.00	59708
fraud	0.89	0.63	0.74	102
accuracy			1.00	59810
macro avg	0.94	0.81	0.87	59810
weighted avg	1.00	1.00	1.00	59810

Macro average

- You give equal importance to all classes and average over all classes.
- For instance, in the example above, recall for non-fraud is 1.0 and fraud is 0.63, and so macro average is 0.81.
- More relevant in case of multi-class problems.

Weighted average

- Weighted by the number of samples in each class.
- Divide by the total number of samples.

Which one is relevant when depends upon whether you think each class should have the same weight or each sample should have the same weight.

Interim summary

- Accuracy is misleading when you have class imbalance.
- A confusion matrix provides a way to break down errors made by our model.
- We looked at three metrics based on confusion matrix:
 - precision, recall, f1-score.

- Note that what you consider "positive" (fraud in our case) is important when calculating precision, recall, and f1-score.
- If you flip what is considered positive or negative, we'll end up with different TP, FP, TN, FN, and hence different precision, recall, and f1-scores.

Evaluation metrics overview

There is a lot of terminology here.

Confusion Matrix Components

Below are different components of a confusion matrix for a binary classification task with classes **Positive** and **Negative**.

		Predicted			# positives
		Positive	Negative	Total	
Actual	Positive	True positive (TP)	False negative (FN) (Type 2 error)		# positives
	Negative	False positive (FP)	True negative (TN) (Type 1 error)		
Total		TP + FP	FN + TN	# examples	

Confusion Matrix Example

		Predicted		Total
		Positive	Negative	
Actual	Positive	80	40	120
	Negative	20	60	80
Total		100	100	200

Accuracy and Error

$$\text{accuracy} = \frac{TP+TN}{TP+FP+TN+FN} = \frac{TP+TN}{\#examples}$$

$$\text{error} = \frac{FP+FN}{TP+FP+TN+FN} = \frac{FP+FN}{\#examples}$$

Examples

$$\text{accuracy} = \frac{80+60}{200} = \frac{140}{200} = 0.70$$

$$\text{error} = \frac{20+40}{200} = \frac{60}{200} = 0.30$$

Precision

$$\text{precision} = \frac{TP}{TP+FP}$$

Example

$$\text{precision} = \frac{80}{100} = 0.80$$

Recall/TP rate/sensitivity

$$\text{recall} = \frac{TP}{TP+FN} = \frac{TP}{\#positives}$$

Example

$$\text{recall} = \frac{80}{120} = 0.666$$

F_1 score

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Example

$$F_1 = 2 \times \frac{0.8 \times 0.666}{0.8 + 0.666} = 0.727$$

True Negative Rate (specificity)

$$\text{tnr} = \frac{TN}{\#negatives}$$

Example

$$\text{specificity} = \frac{60}{80} = 0.75$$

False Positive Rate

$$\text{fpr} = \frac{FP}{FP+TN} = \frac{FP}{\#negatives}$$

Example

$$\text{fpr} = \frac{20}{80} = 0.25$$

False Negative Rate

$$\text{fnr} = \frac{FN}{FN+TP} = \frac{FN}{\#positives}$$

Example

$$\text{fnr} = \frac{40}{120} = 0.333$$

Cross validation with different metrics

- We can pass different evaluation metrics with `scoring` argument of `cross_validate`.

In [125]:

```

1 scoring = [
2     "accuracy",
3     "f1",
4     "recall",
5     "precision",
6 ] # scoring can be a string, a list, or a dictionary
7 pipe = make_pipeline(StandardScaler(), LogisticRegression())
8 scores = cross_validate(
9     pipe, X_train_big, y_train_big, return_train_score=True, scoring=scoring
10 )
11 pd.DataFrame(scores)

```

Out[125]:

	fit_time	score_time	test_accuracy	train_accuracy	test_f1	train_f1	test_recall	train_recall	1
0	1.158515	0.085190	0.999147	0.999367	0.711864	0.783726	0.617647	0.675277	
1	1.274491	0.093204	0.999298	0.999329	0.766667	0.770878	0.676471	0.664207	
2	1.098696	0.093120	0.999273	0.999216	0.743363	0.726477	0.617647	0.612546	
3	1.166864	0.081010	0.999172	0.999279	0.697248	0.753747	0.558824	0.649446	
4	1.238013	0.080994	0.999172	0.999223	0.702703	0.731602	0.582090	0.621324	

- You can also create [your own scoring function \(\[https://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html\]\(https://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html\)\)](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html) and pass it to `cross_validate`.

? ? Questions for you

Questions: decision theory, evaluation metrics

1. In medical diagnosis, false positives are more damaging than false negatives (assume "positive" means the person has a disease, "negative" means they don't). T/F
2. In spam classification, false positives are more damaging than false negatives (assume "positive" means the email is spam, "negative" means they it's not). T/F
3. In the medical diagnosis, high recall is more important than high precision. T/F
4. What's the only metric that does not change if the positive class is changed?
5. If the cost of each mistake is equal independently of the decision, it is more informative to look at the macro average of the metrics. (T/F)

Precision-recall curve and ROC curve

- Confusion matrix provides a detailed break down of the errors made by the model.
- But when creating a confusion matrix, we are using "hard" predictions.
- Most classifiers in `scikit-learn` provide `predict_proba` method (or `decision_function`) which provides degree of certainty about predictions by the classifier.
- Can we explore the degree of uncertainty to understand and improve the model performance?

Let's revisit the classification report on our fraud detection example.

```
In [126]: 1 pipe_lr = make_pipeline(StandardScaler(), LogisticRegression())
2 pipe_lr.fit(X_train, y_train);
```

In [127]:

```
1 y_pred = pipe_lr.predict(X_valid)
2 print(classification_report(y_valid, y_pred, target_names=["non-fraud",
                                                               "fraud"]))

precision    recall   f1-score   support
non-fraud      1.00     1.00     1.00     59708
    fraud       0.89     0.63     0.74      102

accuracy          -         -         -     59810
macro avg       0.94     0.81     0.87     59810
weighted avg     1.00     1.00     1.00     59810
```

By default, predictions use the threshold of 0.5. If `predict_proba` > 0.5, predict "fraud" else predict "non-fraud".

In [128]:

```
1 y_pred = pipe_lr.predict_proba(X_valid)[:, 1] > 0.50
2 print(classification_report(y_valid, y_pred, target_names=["non-fraud",
                                                               "fraud"]))

precision    recall   f1-score   support
non-fraud      1.00     1.00     1.00     59708
    fraud       0.89     0.63     0.74      102

accuracy          -         -         -     59810
macro avg       0.94     0.81     0.87     59810
weighted avg     1.00     1.00     1.00     59810
```

- Suppose for your business it is more costly to miss fraudulent transactions and you want to achieve a recall of at least 75% for the "fraud" class.
- One way to do this is by changing the threshold of `predict_proba`.
 - `predict` returns 1 when `predict_proba`'s probabilities are above 0.5 for the "fraud" class.

Key idea: what if we threshold the probability at a smaller value so that we identify more examples as "fraud" examples?

Let's lower the threshold to 0.1. In other words, predict the examples as "fraud" if `predict_proba` > 0.1.

In [129]:

```
1 y_pred_lower_threshold = pipe_lr.predict_proba(X_valid)[:, 1] > 0.1
```

```
In [130]: 1 print(classification_report(y_valid, y_pred_lower_threshold))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	59708
1	0.78	0.76	0.77	102
accuracy			1.00	59810
macro avg	0.89	0.88	0.89	59810
weighted avg	1.00	1.00	1.00	59810

Operating point

- Now our recall for "fraud" class is ≥ 0.75 .
- Setting a requirement on a classifier (e.g., recall of ≥ 0.75) is called setting the **operating point**.
- It's usually driven by business goals and is useful to make performance guarantees to customers.

Precision/Recall tradeoff

- But there is a trade-off between precision and recall.
- If you identify more things as "fraud", recall is going to increase but there are likely to be more false positives.

Let's sweep through different thresholds.

```
In [131]: 1 thresholds = np.arange(0.0, 1.0, 0.1)
2 thresholds
```

```
Out[131]: array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

```
In [132]: 1 pr_dict = {"threshold": [], "precision": [], "recall": [], "f1 score": []
2 for threshold in thresholds:
3     preds = pipe_lr.predict_proba(X_valid)[:, 1] > threshold
4     pr_dict["threshold"].append(threshold)
5     pr_dict["precision"].append(precision_score(y_valid, preds))
6     pr_dict["recall"].append(recall_score(y_valid, preds))
7     pr_dict["f1 score"].append(f1_score(y_valid, preds))
```

```
In [133]: 1 pd.DataFrame(pr_dict)
```

Out[133]:

	threshold	precision	recall	f1 score
0	0.0	0.001705	1.000000	0.003405
1	0.1	0.780000	0.764706	0.772277
2	0.2	0.795699	0.725490	0.758974
3	0.3	0.819277	0.666667	0.735135
4	0.4	0.876712	0.627451	0.731429
5	0.5	0.888889	0.627451	0.735632
6	0.6	0.897059	0.598039	0.717647
7	0.7	0.892308	0.568627	0.694611
8	0.8	0.901639	0.539216	0.674847
9	0.9	0.894737	0.500000	0.641509

Decreasing the threshold

- Decreasing the threshold means a lower bar for predicting fraud.
 - You are willing to risk more false positives in exchange of more true positives.
 - recall would either stay the same or go up and precision is likely to go down
 - occasionally, precision may increase if all the new examples after decreasing the threshold are TPs.

Increasing the threshold

- Increasing the threshold means a higher bar for predicting fraud.
 - recall would go down or stay the same but precision is likely to go up
 - occasionally, precision may go down as the denominator for precision is TP+FP.

Precision-recall curve

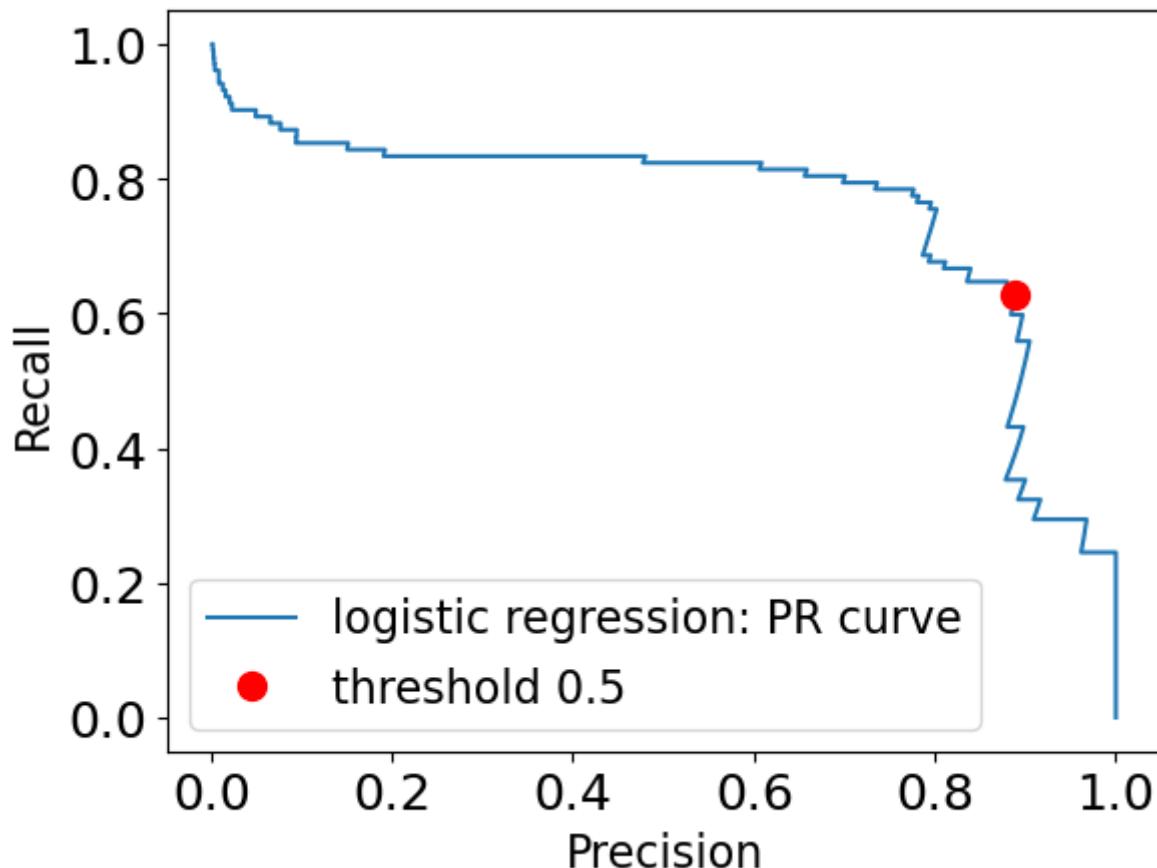
Often, when developing a model, it's not always clear what the operating point will be and to understand the the model better, it's informative to look at all possible thresholds and corresponding trade-offs of precision and recall in a plot.

In [134]:

```

1 from sklearn.metrics import precision_recall_curve
2
3 precision, recall, thresholds = precision_recall_curve(
4     y_valid, pipe_lr.predict_proba(X_valid)[:, 1]
5 )
6 plt.plot(precision, recall, label="logistic regression: PR curve")
7 plt.xlabel("Precision")
8 plt.ylabel("Recall")
9 plt.plot(
10     precision_score(y_valid, pipe_lr.predict(X_valid)),
11     recall_score(y_valid, pipe_lr.predict(X_valid)),
12     "or",
13     markersize=10,
14     label="threshold 0.5",
15 )
16 plt.legend(loc="best");

```



- Each point in the curve corresponds to a possible threshold of the `predict_proba` output.
- We can achieve a recall of 0.8 at a precision of 0.4.
- The red dot marks the point corresponding to the threshold 0.5.
- The top-right would be a perfect classifier (precision = recall = 1).

- The threshold is not shown here, but it's going from 0 (upper-left) to 1 (lower right).
- At a threshold of 0 (upper left), we are classifying everything as "fraud".
- Raising the threshold increases the precision but at the expense of lowering the recall.

- At the extreme right, where the threshold is 1, we get into the situation where all the examples classified as "fraud" are actually "fraud"; we have no false positives.
- Here we have a high precision but lower recall.

A few comments on PR curve

- Different classifiers might work well in different parts of the curve, i.e., at different operating points.
- We can compare PR curves of different classifiers to understand these differences.

AP score

- Often it's useful to have one number summarizing the PR plot (e.g., in hyperparameter optimization)
- One way to do this is by computing the area under the PR curve.
- This is called **average precision** (AP score)
- AP score has a value between 0 (worst) and 1 (best).

In [135]:

```

1 from sklearn.metrics import average_precision_score
2
3 ap_lr = average_precision_score(y_valid, pipe_lr.predict_proba(X_valid))
4 print("Average precision of logistic regression: {:.3f}".format(ap_lr))

```

Average precision of logistic regression: 0.757

AP vs. F1-score

It is very important to note this distinction:

- F1 score is for a given threshold and measures the quality of `predict`.
- AP score is a summary across thresholds and measures the quality of `predict_proba`.

Remember to pick the desired threshold based on the results on the validation set and **not** on the test set.

Receiver Operating Characteristic (ROC) curve

- Another commonly used tool to analyze the behavior of classifiers at different thresholds.
- Similar to PR curve, it considers all possible thresholds for a given classifier given by `predict_proba` but instead of precision and recall it plots false positive rate (FPR) and true

positive rate (TPR, or recall).

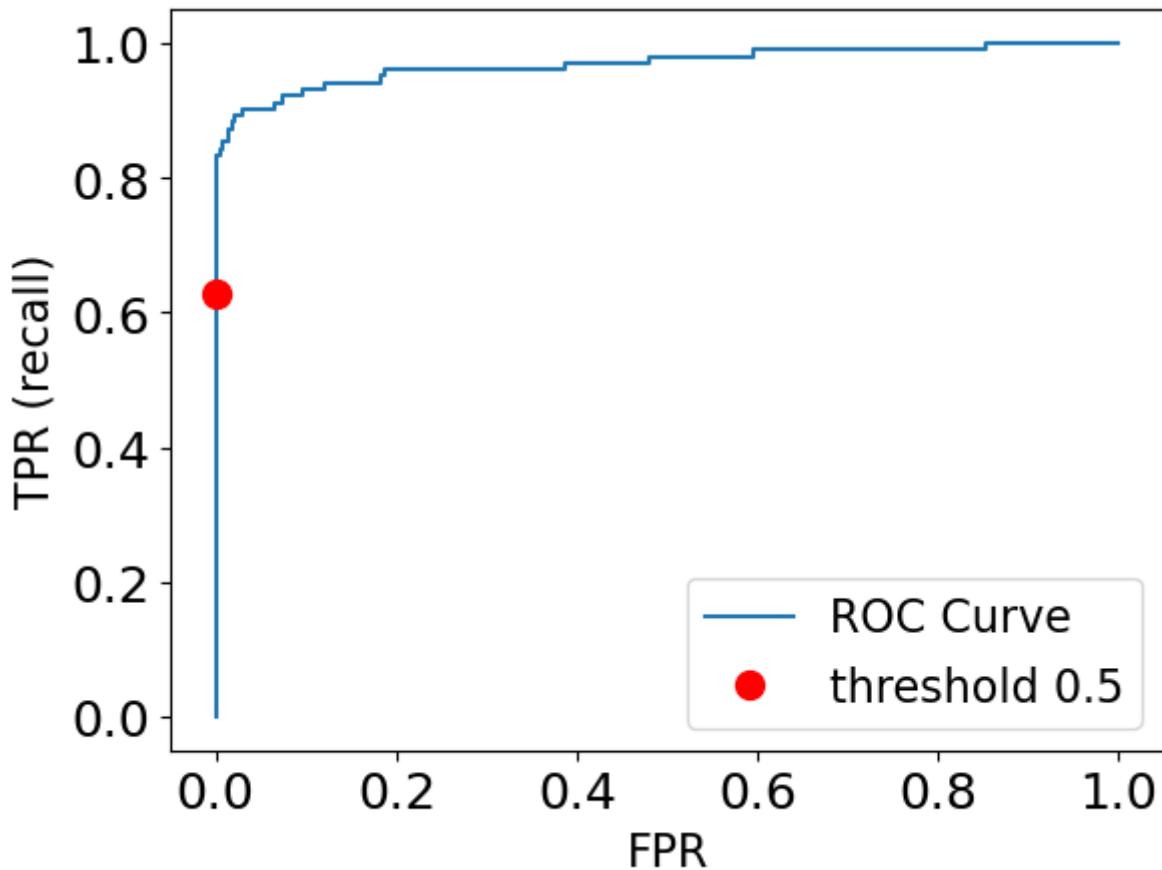
$$FPR = \frac{FP}{FP + TN}, TPR = \frac{TP}{TP + FN}$$

In [136]:

```

1 from sklearn.metrics import roc_curve
2
3 fpr, tpr, thresholds = roc_curve(y_valid, pipe_lr.predict_proba(X_valid)
4 plt.plot(fpr, tpr, label="ROC Curve")
5 plt.xlabel("FPR")
6 plt.ylabel("TPR (recall)")
7
8 default_threshold = np.argmin(np.abs(thresholds - 0.5))
9
10 plt.plot(
11     fpr[default_threshold],
12     tpr[default_threshold],
13     "or",
14     markersize=10,
15     label="threshold 0.5",
16 )
17 plt.legend(loc="best");

```



- The ideal curve is close to the top left
 - Ideally, you want a classifier with high recall while keeping low false positive rate.
- The red dot corresponds to the threshold of 0.5, which is used by predict.

Area under the curve (AUC)

- AUC provides a single meaningful number for the model performance.

In [137]:

```

1 from sklearn.metrics import roc_auc_score
2
3 roc_lr = roc_auc_score(y_valid, pipe_lr.predict_proba(X_valid)[:, 1])
4 print("AUC for logistic regression: {:.3f}".format(roc_lr))

```

AUC for logistic regression: 0.969

- AUC of 0.5 means random chance.
- AUC can be interpreted as evaluating the **ranking** of positive examples.
- What's the probability that a randomly picked positive point has a higher score according to the classifier than a randomly picked point from the negative class.
- AUC of 1.0 means all positive points have a higher score than all negative points.

For classification problems with imbalanced classes, using AP score or AUC is often much more meaningful than using accuracy.

Let's look at all the scores at once

In [138]:

```

1 scoring = ["accuracy", "f1", "recall", "precision", "roc_auc", "average"]
2 pipe = make_pipeline(StandardScaler(), LogisticRegression())
3 scores = cross_validate(pipe, X_train_big, y_train_big, scoring=scoring)
4 pd.DataFrame(scores).mean()

```

Out[138]:

fit_time	1.197308
score_time	0.138669
test_accuracy	0.999212
test_f1	0.724369
test_recall	0.610536
test_precision	0.894228
test_roc_auc	0.967438
test_average_precision	0.744030
dtype:	float64

Dealing with class imbalance

Class imbalance in training sets

- This typically refers to having many more examples of one class than another in one's training set.
- Real world data is often imbalanced.
 - Our Credit Card Fraud dataset is imbalanced.
 - Ad clicking data is usually drastically imbalanced. (Only around ~0.01% ads are clicked.)
 - Spam classification datasets are also usually imbalanced.

Addressing class imbalance

A very important question to ask yourself: "Why do I have a class imbalance?"

- Is it because one class is much more rare than the other?
 - If it's just because one is more rare than the other, you need to ask whether you care about one type of error more than the other.
- Is it because of my data collection methods?
 - If it's the data collection, then that means *your test and training data come from different distributions!*

In some cases, it may be fine to just ignore the class imbalance.

Which type of error is more important?

- False positives (FPs) and false negatives (FNs) have quite different real-world consequences.
- In PR curve and ROC curve, we saw how changing the prediction threshold can change FPs and FNs.
- We can then pick the threshold that's appropriate for our problem.
- Example: if we want high recall, we may use a lower threshold (e.g., a threshold of 0.1). We'll then catch more fraudulent transactions.

```
In [139]: 1 pipe_lr = make_pipeline(StandardScaler(), LogisticRegression())
2 pipe_lr.fit(X_train, y_train)
3 y_pred = pipe_lr.predict(X_valid)
4 print(classification_report(y_valid, y_pred, target_names=["non-fraud",
```

	precision	recall	f1-score	support
non-fraud	1.00	1.00	1.00	59708
fraud	0.89	0.63	0.74	102
accuracy			1.00	59810
macro avg	0.94	0.81	0.87	59810
weighted avg	1.00	1.00	1.00	59810

In [140]:

```

1 y_pred = pipe_lr.predict_proba(X_valid)[:, 1] > 0.10
2 print(classification_report(y_valid, y_pred, target_names=["non-fraud",
                                                               "fraud"]))

```

	precision	recall	f1-score	support
non-fraud	1.00	1.00	1.00	59708
fraud	0.78	0.76	0.77	102
accuracy			1.00	59810
macro avg	0.89	0.88	0.89	59810
weighted avg	1.00	1.00	1.00	59810

Handling imbalance

Can we change the model itself rather than changing the threshold so that it takes into account the errors that are important to us?

There are two common approaches for this:

- **Changing the data (optional)** (not covered in this course)
 - Undersampling
 - Oversampling
 - Random oversampling
 - SMOTE
- **Changing the training procedure**
 - `class_weight`

Changing the training procedure

- All `sklearn` classifiers have a parameter called `class_weight`.
- This allows you to specify that one class is more important than another.
- For example, maybe a false negative is 10x more problematic than a false positive.

Example: `class_weight` parameter of `sklearn LogisticRegression`

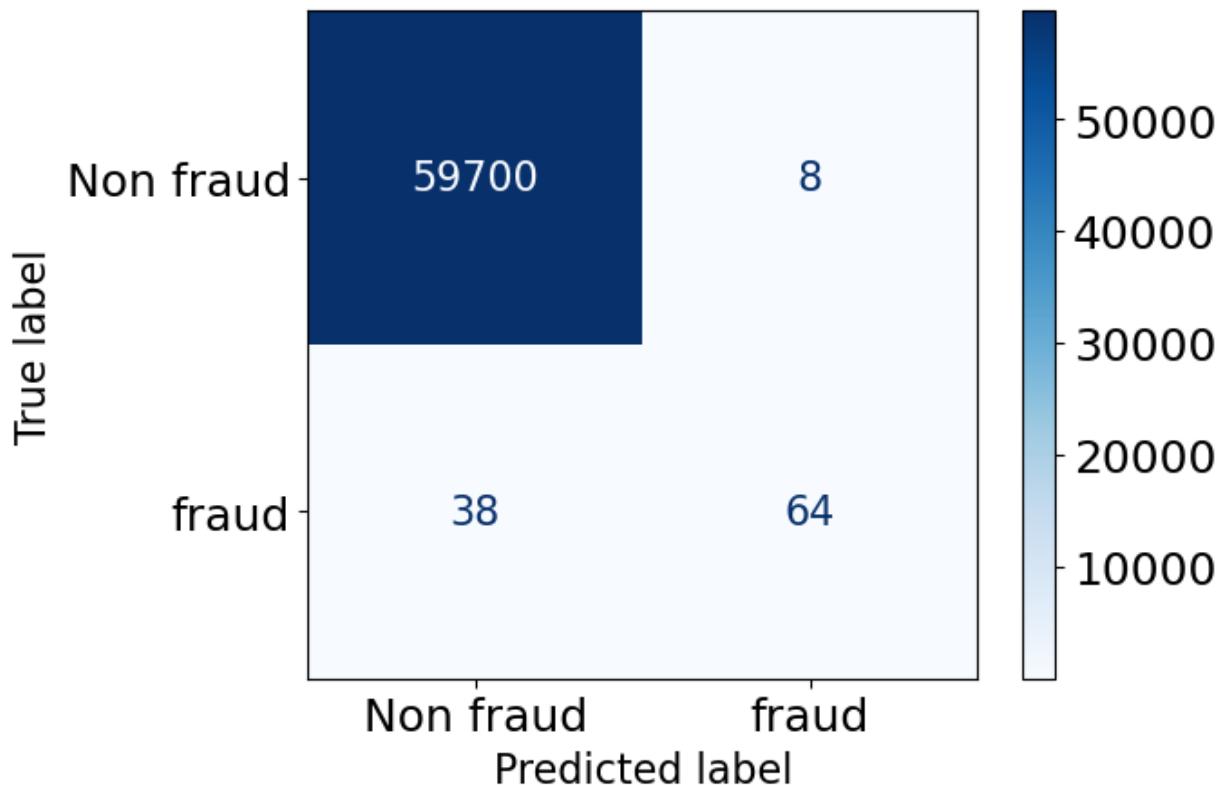
```
class sklearn.linear_model.LogisticRegression(penalty='l2', dual=False, tol=0.0001,
                                              C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None,
                                              random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0,
                                              warm_start=False, n_jobs=None, l1_ratio=None)
```

`class_weight`: dict or 'balanced', default=None

Weights associated with classes in the form {class_label: weight}. If not given, all

```
In [ ]: 1 url = "https://scikit-learn.org/stable/modules/generated/sklearn.linear
2 #HTML("<iframe src=%s width=1000 height=650></iframe>" % url)
3 IPython.display.IFrame(url, width=1000, height=650)
```

```
In [141]: 1 plot_confusion_matrix(
2     pipe_lr,
3     X_valid,
4     y_valid,
5     display_labels=["Non fraud", "fraud"],
6     values_format="d",
7     cmap=plt.cm.Blues,
8 );
```



```
In [142]: 1 pipe_lr.named_steps["logisticregression"].classes_
```

```
Out[142]: array([0, 1])
```

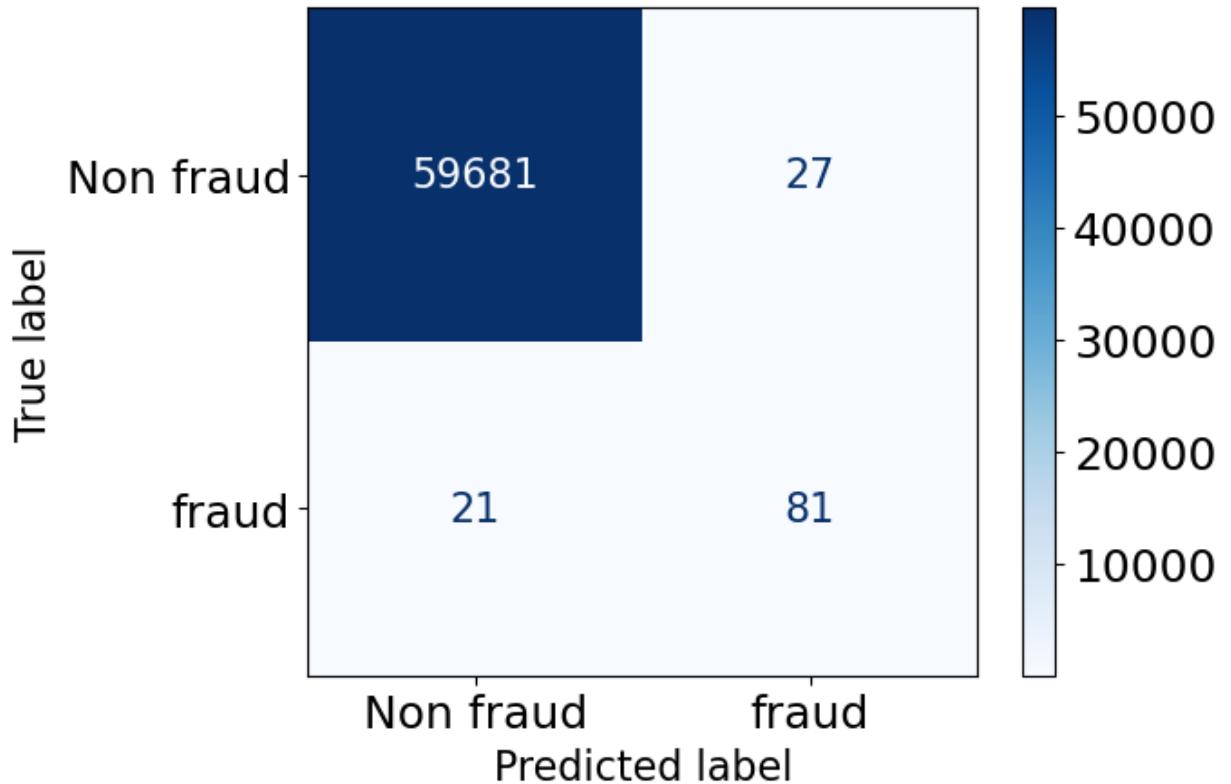
Let's set "fraud" class a weight of 10.

In [143]:

```

1 pipe_lr_weight = make_pipeline(
2     StandardScaler(), LogisticRegression(max_iter=500, class_weight={0:
3 })
4 pipe_lr_weight.fit(X_train, y_train)
5 plot_confusion_matrix(
6     pipe_lr_weight,
7     X_valid,
8     y_valid,
9     display_labels=["Non fraud", "fraud"],
10    values_format="d",
11    cmap=plt.cm.Blues,
12 );

```



- Notice we've reduced false negatives and predicted more Fraud this time.
- This was equivalent to saying give 10x more "importance" to fraud class.
- Note that as a consequence we are also increasing false positives.

`class_weight="balanced"`

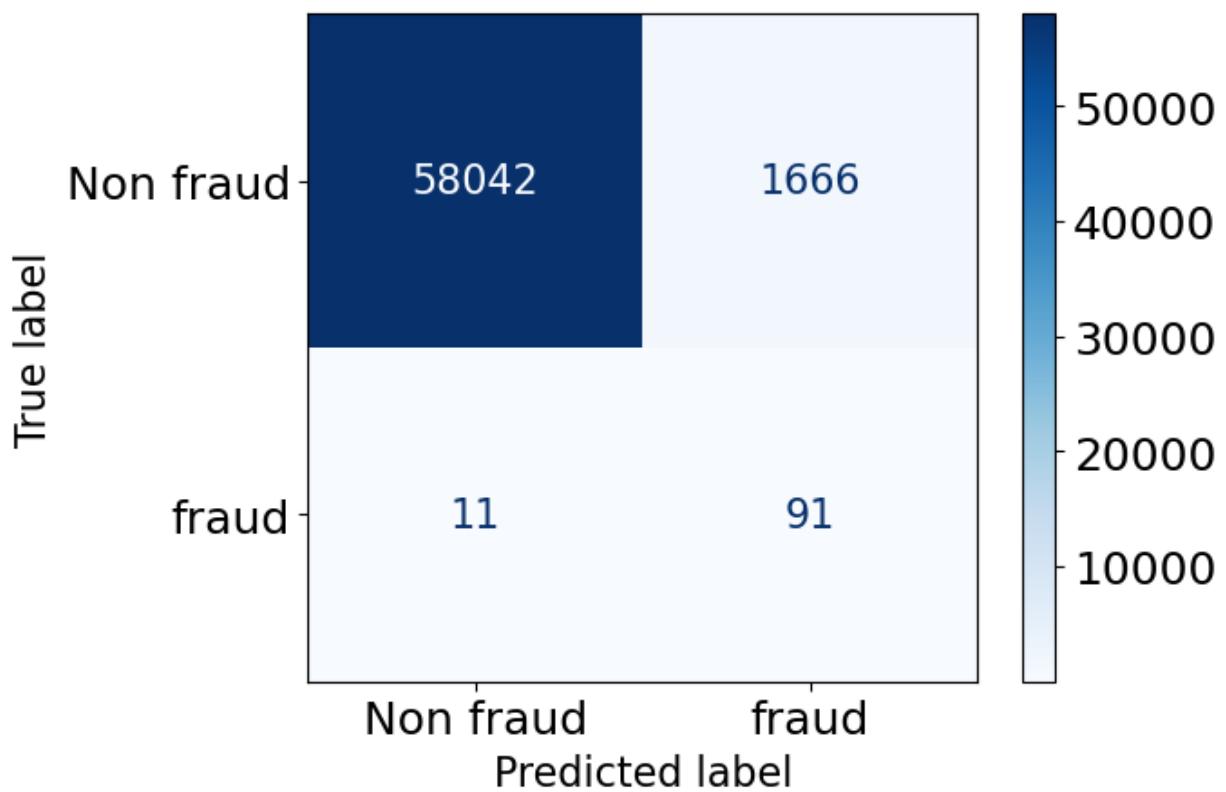
- A useful setting is `class_weight="balanced"` .
- This sets the weights so that the classes are "equal".

`class_weight: dict, 'balanced' or None` If 'balanced', class weights will be given by $n_{samples} / (n_{classes} * np.bincount(y))$. If a dictionary is given, keys are classes and values are corresponding class weights. If None is given, the class weights will be uniform.

```
sklearn.utils.class_weight.compute_class_weight(class_weight, classes, y)
```

In [144]:

```
1 pipe_lr_balanced = make_pipeline(
2     StandardScaler(), LogisticRegression(max_iter=500, class_weight="balanced")
3 )
4 pipe_lr_balanced.fit(X_train, y_train)
5 plot_confusion_matrix(
6     pipe_lr_balanced,
7     X_valid,
8     y_valid,
9     display_labels=["Non fraud", "fraud"],
10    values_format="d",
11    cmap=plt.cm.Blues,
12 );
```



We have reduced false negatives but we have many more false positives now ...

Are we doing better with `class_weight="balanced"` ?

In [145]:

```
1 comp_dict = {}
2 pipe_lr = make_pipeline(StandardScaler(), LogisticRegression(max_iter=500))
3 scoring = ["accuracy", "f1", "recall", "precision", "roc_auc", "average"]
4 orig_scores = cross_validate(pipe_lr, X_train_big, y_train_big, scoring)
```

In [146]:

```

1 pipe_lr_balanced = make_pipeline(
2     StandardScaler(), LogisticRegression(max_iter=500, class_weight="ba
3 )
4 scoring = ["accuracy", "f1", "recall", "precision", "roc_auc", "average
5 bal_scores = cross_validate(pipe_lr_balanced, X_train_big, y_train_big,
6 comp_dict = {
7     "Original": pd.DataFrame(orig_scores).mean().tolist(),
8     "class_weight='balanced)": pd.DataFrame(bal_scores).mean().tolist()
9 }

```

In [147]:

```
1 pd.DataFrame(comp_dict, index=bal_scores.keys())
```

Out[147]:

	Original	class_weight='balanced'
fit_time	1.179936	1.211615
score_time	0.123018	0.130661
test_accuracy	0.999212	0.973626
test_f1	0.724369	0.103831
test_recall	0.610536	0.896883
test_precision	0.894228	0.055119
test_roc_auc	0.967438	0.970881
test_average_precision	0.744030	0.730627

- Recall is much better but precision has dropped a lot; we have many false positives.
- You could also optimize `class_weight` using hyperparameter optimization for your specific problem.

- Changing the class weight will **generally reduce accuracy**.
- The original model was trying to maximize accuracy.
- Now you're telling it to do something different.
- But that can be fine, accuracy isn't the only metric that matters.

Stratified Splits

- A similar idea of "balancing" classes can be applied to data splits.
- We have the same option in `train_test_split` with the `stratify` argument.
- By default it splits the data so that if we have 10% negative examples in total, then each split will have 10% negative examples.

- If you are carrying out cross validation using `cross_validate`, by default it uses `StratifiedKFold` (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html). From the documentation:

This cross-validation object is a variation of KFold that returns stratified folds. The folds are made by preserving the percentage of samples for each class.

- In other words, if we have 10% negative examples in total, then each fold will have 10% negative examples.

Is stratifying a good idea?

- Well, it's no longer a random sample, which is probably theoretically bad, but not that big of a deal.
- If you have many examples, it shouldn't matter as much.
- It can be especially useful in multi-class, say if you have one class with very few cases.
- In general, these are difficult questions.

Exercise

I have a dataset with 10100 samples, 100 of which belong to the positive class. Which syntax is giving more weight to the minority class?

```
In [ ]: 1 LogisticRegression(max_iter=500, class_weight="balanced")
2 0:100 1:1
3
4 # or
5
6 LogisticRegression(max_iter=500, class_weight={0:1, 1:1000})
```

What did we learn today?

- A number of possible ways to evaluate machine learning models
 - Choose the evaluation metric that makes most sense in your context or which is most common in your discipline
- Two kinds of binary classification problems
 - Distinguishing between two classes (e.g., dogs vs. cats)
 - Spotting a class (e.g., spot fraud transaction, spot spam)

- Precision, recall, f1-score are useful when dealing with spotting problems.
- The thing that we are interested in spotting is considered "positive".
- Do you need to deal with class imbalance in the given problem?
- Methods to deal with class imbalance
 - Changing the training procedure

- class_weight

Relevant papers and resources

- [The Relationship Between Precision-Recall and ROC Curves](https://www.biostat.wisc.edu/~page/rocpr.pdf)
(<https://www.biostat.wisc.edu/~page/rocpr.pdf>)
- [Article claiming that PR curve are better than ROC for imbalanced datasets](https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0118432)
(<https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0118432>)
- [Precision-Recall-Gain Curves: PR Analysis Done Right](https://papers.nips.cc/paper/2015/file/33e8075e9970de0cfea955afd4644bb2-Paper.pdf)
(<https://papers.nips.cc/paper/2015/file/33e8075e9970de0cfea955afd4644bb2-Paper.pdf>)
- [ROC animation](https://github.com/dariyasydykova/open_projects/tree/master/ROC_animation)
(https://github.com/dariyasydykova/open_projects/tree/master/ROC_animation)
- [Generalization in Adaptive Data Analysis and Holdout Reuse](https://arxiv.org/pdf/1506.02629.pdf)
(<https://arxiv.org/pdf/1506.02629.pdf>)

In []:

1

CPSC 330

Applied Machine Learning

Lecture 10: Regression Evaluation Metrics

UBC 2022-23

Instructor: Mathias Lécuyer

Announcement

- Midterm next week, on Wednesday, Feb. 15, from 7:00pm to 8:15pm:
 - CPSC 330 Section 202 students (you) will write the exam in ESB 1012 (<https://learningspaces.ubc.ca/classrooms/esb-1012>) (<https://learningspaces.ubc.ca/classrooms/esb-1012>):
 - More details on [piazza](https://piazza.com/class/lcg06c2ncl06el/post/283) (<https://piazza.com/class/lcg06c2ncl06el/post/283>) (<https://piazza.com/class/lcg06c2ncl06el/post/283>).
- There is a [piazza poll](https://piazza.com/class/lcg06c2ncl06el/post/316) (<https://piazza.com/class/lcg06c2ncl06el/post/316>) for topics to cover in the review session (next Tuesday, Feb 14): <https://piazza.com/class/lcg06c2ncl06el/post/316> (<https://piazza.com/class/lcg06c2ncl06el/post/316>).

Imports

In [41]:

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 import pandas as pd
4 from sklearn.compose import (
5     ColumnTransformer,
6     TransformedTargetRegressor,
7     make_column_transformer,
8 )
9 from sklearn.dummy import DummyRegressor
10 from sklearn.ensemble import RandomForestRegressor
11 from sklearn.impute import SimpleImputer
12 from sklearn.linear_model import LinearRegression, Ridge, RidgeCV
13 from sklearn.metrics import make_scorer, mean_squared_error, r2_score
14 from sklearn.model_selection import cross_val_score, cross_validate, train_test_split
15 from sklearn.pipeline import Pipeline, make_pipeline
16 from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
17 from sklearn.tree import DecisionTreeRegressor
18
19 %matplotlib inline

```

In [42]:

```

1 import warnings
2
3 warnings.simplefilter(action="ignore", category=FutureWarning)

```

Some clarifications on last lecture

- Formula for false positive rate: Fraction of false positives out of all negative examples:

$$FPR = \frac{FP}{FP + TN} = \frac{FP}{N}$$

- This was in context of the ROC curve, which is TPR (a.k.a recall, a.k.a sensitivity) as function of FPR:

$$TPR = \frac{TP}{TP + FN} = \frac{TP}{P}, \quad FPR = \frac{FP}{FP + TN}$$

- Recall is also called sensitivity:

$$\text{Recall} = \text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{P}$$

- Note that TPR/recall/sensitivity is about the positive class (how much of it we find), while FPR is really about the negative class (how much of it we mispredict).
- Precision, recall, f1 score, are only about the positive label:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{P}$$

Processing math: 100%

- The positive class is assumed to be the class label 1 by default. This is configurable through the `pos_label` parameter.

Learning outcomes

From this lecture, students are expected to be able to:

- Carry out feature transformations on somewhat complicated dataset.
- Visualize transformed features as a dataframe.
- Use `Ridge` and `RidgeCV`.
- Explain how `alpha` hyperparameter of `Ridge` relates to the fundamental tradeoff.
- Examine coefficients of transformed features.
- Appropriately select a scoring metric given a regression problem.
- Interpret and communicate the meanings of different scoring metrics on regression problems.
 - MSE, RMSE, R^2 , MAPE
- Apply log-transform on the target values in a regression problem with `TransformedTargetRegressor`.

Dataset

In this lecture, we'll be using [Kaggle House Prices dataset \(<https://www.kaggle.com/c/home-data-for-ml-course>\)](https://www.kaggle.com/c/home-data-for-ml-course). As usual, to run this notebook you'll need to download the data. For this dataset, train and test have already been separated. We'll be working with the train portion in this lecture.

```
In [43]: 1 df = pd.read_csv("../data/housing-kaggle/train.csv")
2 train_df, test_df = train_test_split(df, test_size=0.10, random_state=1)
3 train_df.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	L
302	303	20	RL	118.0	13704	Pave	NaN	IR1		Lvl
767	768	50	RL	75.0	12508	Pave	NaN	IR1		Lvl
429	430	20	RL	130.0	11457	Pave	NaN	IR1		Lvl
1139	1140	30	RL	98.0	8731	Pave	NaN	IR1		Lvl
558	559	60	RL	57.0	21872	Pave	NaN	IR2		HLS

5 rows × 81 columns

- The supervised machine learning problem is predicting housing price given features associated with properties.
- Here, the target is `SalePrice`, which is continuous. So it's a **regression problem** (as opposed to classification).

In [44]: 1 train_df.shape

Out[44]: (1314, 81)

Let's separate x and y

In [45]:

```
1 x_train = train_df.drop(columns=["SalePrice"])
2 y_train = train_df["SalePrice"]
3
4 X_test = test_df.drop(columns=["SalePrice"])
5 y_test = test_df["SalePrice"]
```

EDA

In [46]: 1 train_df.describe()

Out[46]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBui
count	1314.000000	1314.000000	1089.000000	1314.000000	1314.000000	1314.000000	1314.000000
mean	734.182648	56.472603	69.641873	10273.261035	6.076104	5.570015	1970.99543
std	422.224662	42.036646	23.031794	8997.895541	1.392612	1.112848	30.19812
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.00000
25%	369.250000	20.000000	59.000000	7500.000000	5.000000	5.000000	1953.00000
50%	735.500000	50.000000	69.000000	9391.000000	6.000000	5.000000	1972.00000
75%	1099.750000	70.000000	80.000000	11509.000000	7.000000	6.000000	2000.00000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.00000

8 rows × 38 columns

```
In [47]: 1 train_df.info()
```

Processing math: 100%

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1314 entries, 302 to 1389
Data columns (total 81 columns):
 #   Column           Non-Null Count Dtype  
 --- 
 0   Id               1314 non-null   int64  
 1   MSSubClass        1314 non-null   int64  
 2   MSZoning          1314 non-null   object  
 3   LotFrontage       1089 non-null   float64 
 4   LotArea            1314 non-null   int64  
 5   Street             1314 non-null   object  
 6   Alley              81 non-null    object  
 7   LotShape            1314 non-null   object  
 8   LandContour         1314 non-null   object  
 9   Utilities           1314 non-null   object  
 10  LotConfig           1314 non-null   object  
 11  LandSlope           1314 non-null   object  
 12  Neighborhood        1314 non-null   object  
 13  Condition1          1314 non-null   object  
 14  Condition2          1314 non-null   object  
 15  BldgType            1314 non-null   object  
 16  HouseStyle          1314 non-null   object  
 17  OverallQual         1314 non-null   int64  
 18  OverallCond         1314 non-null   int64  
 19  YearBuilt            1314 non-null   int64  
 20  YearRemodAdd        1314 non-null   int64  
 21  RoofStyle            1314 non-null   object  
 22  RoofMatl             1314 non-null   object  
 23  Exterior1st          1314 non-null   object  
 24  Exterior2nd          1314 non-null   object  
 25  MasVnrType           1307 non-null   object  
 26  MasVnrArea           1307 non-null   float64 
 27  ExterQual            1314 non-null   object  
 28  ExterCond            1314 non-null   object  
 29  Foundation           1314 non-null   object  
 30  BsmtQual             1280 non-null   object  
 31  BsmtCond             1280 non-null   object  
 32  BsmtExposure         1279 non-null   object  
 33  BsmtFinType1          1280 non-null   object  
 34  BsmtFinSF1            1314 non-null   int64  
 35  BsmtFinType2          1280 non-null   object  
 36  BsmtFinSF2            1314 non-null   int64  
 37  BsmtUnfSF             1314 non-null   int64  
 38  TotalBsmtSF           1314 non-null   int64  
 39  Heating                1314 non-null   object  
 40  HeatingQC              1314 non-null   object  
 41  CentralAir             1314 non-null   object  
 42  Electrical              1313 non-null   object  
 43  1stFlrSF                1314 non-null   int64  
 44  2ndFlrSF                1314 non-null   int64  
 45  LowQualFinSF             1314 non-null   int64  
 46  GrLivArea                1314 non-null   int64  
 47  BsmtFullBath             1314 non-null   int64  
 48  BsmtHalfBath             1314 non-null   int64  
 49  FullBath                  1314 non-null   int64  
 50  HalfBath                  1314 non-null   int64  
 51  BedroomAbvGr              1314 non-null   int64
```

Processing math: 100%

```
52 KitchenAbvGr    1314 non-null    int64
53 KitchenQual     1314 non-null    object
54 TotRmsAbvGrd   1314 non-null    int64
55 Functional      1314 non-null    object
56 Fireplaces       1314 non-null    int64
57 FireplaceQu     687 non-null    object
58 GarageType       1241 non-null    object
59 GarageYrBlt     1241 non-null    float64
60 GarageFinish    1241 non-null    object
61 GarageCars       1314 non-null    int64
62 GarageArea       1314 non-null    int64
63 GarageQual      1241 non-null    object
64 GarageCond      1241 non-null    object
65 PavedDrive      1314 non-null    object
66 WoodDeckSF      1314 non-null    int64
67 OpenPorchSF     1314 non-null    int64
68 EnclosedPorch   1314 non-null    int64
69 3SsnPorch        1314 non-null    int64
70 ScreenPorch     1314 non-null    int64
71 PoolArea         1314 non-null    int64
72 PoolQC          7 non-null     object
73 Fence            259 non-null    object
74 MiscFeature      50 non-null     object
75 MiscVal          1314 non-null    int64
76 MoSold           1314 non-null    int64
77 YrSold           1314 non-null    int64
78 SaleType          1314 non-null    object
79 SaleCondition    1314 non-null    object
80 SalePrice         1314 non-null    int64
dtypes: float64(3), int64(35), object(43)
memory usage: 841.8+ KB
```

pandas_profiler

We do not have `pandas_profiling` in our course environment. You will have to install it in the environment on your own if you want to run the code below.

```
conda install -c conda-forge pandas-profiling
```

In [48]:

```
1 from pandas_profiling import ProfileReport
2
3 #profile = ProfileReport(train_df, title="Pandas Profiling Report") #
4 #profile.to_notebook_iframe()
```

Processing math: 100%

Feature types

- Do not blindly trust all the info given to you by automated tools.
- How does pandas profiling figure out the data type?
 - You can look at the Python data type and say floats are numeric, strings are categorical.
 - However, in doing so you would miss out on various subtleties such as some of the string features being ordinal rather than truly categorical.
 - Also, it will think free text is categorical.

- In addition to tools such as above, it's important to go through data description to understand the data.
- The data description for our dataset is available [here \(\[https://www.kaggle.com/c/home-data-for-ml-course/data?select=data_description.txt\]\(https://www.kaggle.com/c/home-data-for-ml-course/data?select=data_description.txt\)\)](https://www.kaggle.com/c/home-data-for-ml-course/data?select=data_description.txt).

Feature types

- We have mixed feature types and a bunch of missing values.
- Now, let's identify feature types and transformations.

- Let's get the numeric-looking columns.

```
In [49]: 1 numeric_looking_columns = X_train.select_dtypes(include=np.number).columns
          2 print(numeric_looking_columns)
```

['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold']

Not all numeric looking columns are necessarily numeric.

```
In [50]: 1 train_df["MSSubClass"].unique()
```

```
Out[50]: array([ 20,  50,  30,  60, 160,  85,  90, 120, 180,  80,  70,  75, 190,
        45,  40])
```

MSSubClass: Identifies the type of dwelling involved in the sale.

Processing math: 100%

```
20  1-STORY 1946 & NEWER ALL STYLES
30  1-STORY 1945 & OLDER
40  1-STORY W/FINISHED ATTIC ALL AGES
45  1-1/2 STORY - UNFINISHED ALL AGES
50  1-1/2 STORY FINISHED ALL AGES
60  2-STORY 1946 & NEWER
70  2-STORY 1945 & OLDER
75  2-1/2 STORY ALL AGES
80  SPLIT OR MULTI-LEVEL
85  SPLIT FOYER
90  DUPLEX - ALL STYLES AND AGES
120 1-STORY PUD (Planned Unit Development) - 1946 & NEWER
```

Also, month sold is more of a categorical feature than a numeric feature.

```
In [51]: 1 train_df[ "MoSold" ].unique() # Month Sold
```

```
Out[51]: array([ 1,  7,  3,  5,  8, 10,  6,  9, 12,  2,  4, 11])
```

Processing math: 100%

In [52]:

```

1 drop_features = ["Id"]
2 numeric_features = [
3     "BedroomAbvGr",
4     "KitchenAbvGr",
5     "LotFrontage",
6     "LotArea",
7     "OverallQual",
8     "OverallCond",
9     "YearBuilt",
10    "YearRemodAdd",
11    "MasVnrArea",
12    "BsmtFinSF1",
13    "BsmtFinSF2",
14    "BsmtUnfSF",
15    "TotalBsmtSF",
16    "1stFlrSF",
17    "2ndFlrSF",
18    "LowQualFinSF",
19    "GrLivArea",
20    "BsmtFullBath",
21    "BsmtHalfBath",
22    "FullBath",
23    "HalfBath",
24    "TotRmsAbvGrd",
25    "Fireplaces",
26    "GarageYrBlt",
27    "GarageCars",
28    "GarageArea",
29    "WoodDeckSF",
30    "OpenPorchSF",
31    "EnclosedPorch",
32    "3SsnPorch",
33    "ScreenPorch",
34    "PoolArea",
35    "MiscVal",
36    "YrSold",
37 ]

```

I've not looked at all the features carefully. It might be appropriate to apply some other encoding on some of the numeric features above.

In [53]:

```
1 set(numeric_looking_columns) - set(numeric_features) - set(drop_feature)
```

Out[53]:

```
{'MSSubClass', 'MoSold'}
```

We'll treat the above numeric-looking features as categorical features.

- There are a bunch of ordinal features in this dataset.
- Ordinal features with the same scale

Processing math: 100%

- Poor (Po), Fair (Fa), Typical (Ta), Good (Gd), Excellent (Ex)

- These we'll be calling `ordinal_features_reg`.
- Ordinal features with different scales
 - These we'll be calling `ordinal_features_oth`.

```
In [54]: 1 ordinal_features_reg = [
2     "ExterQual",
3     "ExterCond",
4     "BsmtQual",
5     "BsmtCond",
6     "HeatingQC",
7     "KitchenQual",
8     "FireplaceQu",
9     "GarageQual",
10    "GarageCond",
11    "PoolQC",
12 ]
13 ordering = [
14     "Po",
15     "Fa",
16     "TA",
17     "Gd",
18     "Ex",
19 ] # if N/A it will just impute something, per below
20 ordering_ordinal_reg = [ordering] * len(ordinal_features_reg)
21 ordering_ordinal_reg
```

```
Out[54]: [[ 'Po', 'Fa', 'TA', 'Gd', 'Ex'],
 [ 'Po', 'Fa', 'TA', 'Gd', 'Ex']]
```

We'll pass the above as categories in our `OrdinalEncoder`.

- There are a bunch more ordinal features using different scales.
 - These we'll be calling `ordinal_features_oth`.
 - We are encoding them separately.

In [55]:

```
1 ordinal_features_oth = [
2     "BsmtExposure",
3     "BsmtFinType1",
4     "BsmtFinType2",
5     "Functional",
6     "Fence",
7 ]
8 ordering_ordinal_oth = [
9     ['NA', 'No', 'Mn', 'Av', 'Gd'],
10    ['NA', 'Unf', 'LwQ', 'Rec', 'BLQ', 'ALQ', 'GLQ'],
11    ['NA', 'Unf', 'LwQ', 'Rec', 'BLQ', 'ALQ', 'GLQ'],
12    ['Sal', 'Sev', 'Maj2', 'Maj1', 'Mod', 'Min2', 'Min1', 'Typ'],
13    ['NA', 'MnWw', 'GdWo', 'MnPrv', 'GdPrv']
14 ]
```

The remaining features are categorical features.

Processing math: 100%

In [56]:

```

1 categorical_features = list(
2     set(X_train.columns)
3     - set(numeric_features)
4     - set(ordinal_features_reg)
5     - set(ordinal_features_oth)
6     - set(drop_features)
7 )
8 categorical_features

```

Out[56]:

```
[ 'MasVnrType',
  'Neighborhood',
  'Condition2',
  'Alley',
  'SaleCondition',
  'Electrical',
  'HouseStyle',
  'GarageFinish',
  'RoofStyle',
  'MoSold',
  'Exterior1st',
  'Street',
  'MSSubClass',
  'LotShape',
  'PavedDrive',
  'Heating',
  'SaleType',
  'Utilities',
  'GarageType',
  'BldgType',
  'MiscFeature',
  'LotConfig',
  'CentralAir',
  'LandSlope',
  'Condition1',
  'Exterior2nd',
  'MSZoning',
  'LandContour',
  'RoofMatl',
  'Foundation' ]
```

- We are not doing it here but we can engineer our own features too.
- Would price per square foot be a good feature to add in here?

Applying feature transformations

- Since we have mixed feature types, let's use `ColumnTransformer` to apply different transformations on different features types.

In [57]:

```
1 from sklearn.compose import ColumnTransformer, make_column_transformer
2
3 numeric_transformer = make_pipeline(SimpleImputer(strategy="median"), S
4 ordinal_transformer_reg = make_pipeline(
5     SimpleImputer(strategy="most_frequent"),
6     OrdinalEncoder(categories=ordering_ordinal_reg),
7 )
8
9 ordinal_transformer_oth = make_pipeline(
10    SimpleImputer(strategy="most_frequent"),
11    OrdinalEncoder(categories=ordering_ordinal_oth),
12 )
13
14 categorical_transformer = make_pipeline(
15    SimpleImputer(strategy="constant", fill_value="missing"),
16    OneHotEncoder(handle_unknown="ignore", sparse=False),
17 )
18
19 preprocessor = make_column_transformer(
20     ("drop", drop_features),
21     (numeric_transformer, numeric_features),
22     (ordinal_transformer_reg, ordinal_features_reg),
23     (ordinal_transformer_oth, ordinal_features_oth),
24     (categorical_transformer, categorical_features),
25 )
```

Processing math: 100%

Examining the preprocessed data

```
In [58]: 1  preprocessor.fit(X_train) # Calling fit to examine all the transformers  
2  preprocessor.named_transformers_
```

Processing math: 100%

Processing math: 100%

In [59]:

```

1 ohe_columns = list(
2     preprocessor.named_transformers_[ "pipeline-4" ]
3     .named_steps[ "onehotencoder" ]
4     .get_feature_names(categorical_features)
5 )
6 new_columns = numeric_features + ordinal_features_reg + ordinal_feature

```

In [60]:

```

1 X_train_enc = pd.DataFrame(
2     preprocessor.transform(X_train), index=X_train.index, columns=new_c
3 )
4 X_train_enc.head()

```

Out[60]:

	BedroomAbvGr	KitchenAbvGr	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	Y
302	0.154795	-0.222647	2.312501	0.381428	0.663680	-0.512408	0.993969	
767	1.372763	-0.222647	0.260890	0.248457	-0.054669	1.285467	-1.026793	
429	0.154795	-0.222647	2.885044	0.131607	-0.054669	-0.512408	0.563314	
1139	0.154795	-0.222647	1.358264	-0.171468	-0.773017	-0.512408	-1.689338	
558	0.154795	-0.222647	-0.597924	1.289541	0.663680	-0.512408	0.828332	

5 rows × 263 columns

In [61]:

```
1 X_train.shape
```

Out[61]:

(1314, 80)

In [62]:

```
1 X_train_enc.shape
```

Out[62]:

(1314, 263)

We went from 80 features to 263 features!!

Other possible preprocessing?

- There is a lot of room for improvement ...
- We're just using `SimpleImputer` .
 - In reality we'd want to go through this more carefully.
 - We may also want to drop some columns that are almost entirely missing.
- We could also check for outliers, and do other exploratory data analysis (EDA).
- But for now this is good enough ...

Model building

DummyRegressor

```
In [63]: 1 dummy = DummyRegressor()
2 pd.DataFrame(cross_validate(dummy, X_train, y_train, cv=10, return_train
```

Out[63]:

	fit_time	score_time	test_score	train_score
0	0.001767	0.000557	-0.003547	0.0
1	0.001569	0.000441	-0.001266	0.0
2	0.001287	0.000450	-0.011767	0.0
3	0.002081	0.000792	-0.006744	0.0
4	0.001895	0.000488	-0.076533	0.0
5	0.001080	0.000369	-0.003133	0.0
6	0.001010	0.000363	-0.000397	0.0
7	0.000984	0.000360	-0.003785	0.0
8	0.001093	0.000494	-0.001740	0.0
9	0.001242	0.000619	-0.000117	0.0

Apply Ridge

- Recall that we are going to use `Ridge()` instead of `LinearRegression()` in this course.
 - It has a hyperparameter `alpha` which controls the fundamental tradeoff.

```
In [64]: 1 lr_pipe = make_pipeline(preprocessor, Ridge())
2 pd.DataFrame(cross_validate(lr_pipe, X_train, y_train, cv=10, return_train
```

Out[64]:

	fit_time	score_time	test_score	train_score
0	0.067918	0.020969	0.861355	0.911906
1	0.073782	0.020503	0.812301	0.913861
2	0.075481	0.021917	0.775283	0.915963
3	0.077557	0.021604	0.874519	0.910849
4	0.074957	0.021669	0.851969	0.911622
5	0.074832	0.024669	0.826198	0.910176
6	0.074639	0.022231	0.825533	0.913781
7	0.074474	0.021569	0.872238	0.910071
8	0.077648	0.024597	0.196663	0.921448
Processing math: 100% 0.072799	0.023196	0.890474	0.908221	

- Quite a bit of variation in the test scores.
- Performing poorly in fold 8. Not sure why.

Tuning the alpha hyperparameter of Ridge

- Recall that Ridge has a hyperparameter `alpha` that controls the fundamental tradeoff.
- This is like `C` in `LogisticRegression` but, annoyingly, `alpha` is the opposite of `C`: large `C` is like small `alpha` and vice versa.
- Smaller `alpha`: more complex model, more variance, lower training error (overfitting)

In [65]:

```

1 alphas = 10.0 ** np.arange(-5, 5, 1)
2 train_scores = []
3 cv_scores = []
4 for alpha in alphas:
5     lr = make_pipeline(preprocessor, Ridge(alpha=alpha))
6     results = cross_validate(lr, X_train, y_train, return_train_score=True)
7     train_scores.append(np.mean(results["train_score"]))
8     cv_scores.append(np.mean(results["test_score"]))

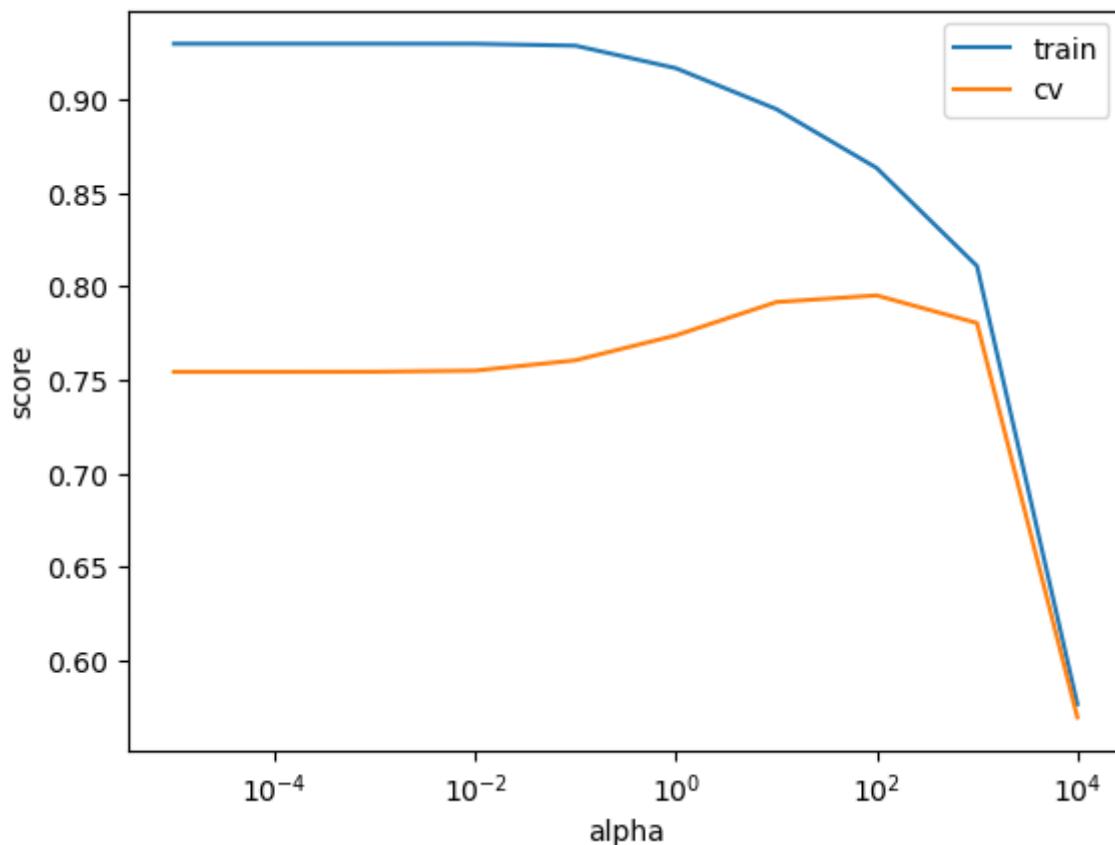
```

In [66]:

```

1 plt.semilogx(alphas, train_scores, label="train")
2 plt.semilogx(alphas, cv_scores, label="cv")
3 plt.legend()
4 plt.xlabel("alpha")
5 plt.ylabel("score");

```



Processing math: 100%

```
In [67]: 1 best_alpha = alphas[np.argmax(cv_scores)]
2 best_alpha
```

Out[67]: 100.0

- It seems alpha=100 is the best choice here.
- General intuition: larger alpha leads to smaller coefficients.
- Smaller coefficients mean the predictions are less sensitive to changes in the data.
- Hence less chance of overfitting (seeing big dependencies when you shouldn't).

RidgeCV

BTW, because it's so common to want to tune alpha with Ridge, sklearn provides a class called RidgeCV, which automatically tunes alpha based on cross-validation.

```
In [68]: 1 ridgecv_pipe = make_pipeline(preprocessor, RidgeCV(alphas=alphas, cv=10)
2 ridgecv_pipe.fit(X_train, y_train);
```

```
In [69]: 1 best_alpha = ridgecv_pipe.named_steps['ridgecv'].alpha_
2 best_alpha
```

Out[69]: 100.0

Let's examine the coefficients

```
In [70]: 1 lr_tuned = make_pipeline(preprocessor, Ridge(alpha=best_alpha))
2 lr_tuned.fit(X_train, y_train)
3 lr_preds = lr_tuned.predict(X_test)
4 lr_preds[:10]
```

```
Out[70]: array([228728.1963872 , 104718.39905565, 155778.96723311, 246316.7111903
1,
127633.10676873, 243207.19441128, 304930.24461291, 145374.5943529
5,
157059.38983893, 128487.51979632])
```

```
In [71]: 1 lr_preds.max(), lr_preds.min()
```

Out[71]: (390726.10647423274, 30791.092505420726)

Let's get the feature names of the transformed data to interpret coefficients.

In [72]:

```

1 ohe_columns = list(
2     preprocessor.named_transformers_[ "pipeline-4" ]
3     .named_steps[ "onehotencoder" ]
4     .get_feature_names(categorical_features)
5 )
6 new_columns = numeric_features + ordinal_features_reg + ordinal_feature
7

```

In [73]:

```

1 df = pd.DataFrame(
2     data={
3         "features": new_columns,
4         "coefficients": lr_tuned.named_steps[ "ridge" ].coef_,
5     }
6 )

```

In [74]:

```
1 df.sort_values( "coefficients" , ascending=False)
```

Out[74]:

	features	coefficients
4	OverallQual	14484.902165
16	GrLivArea	11704.053037
70	Neighborhood_NridgHt	9662.969631
69	Neighborhood_NoRidge	9497.598615
36	BsmtQual	8073.088562
...
249	RoofMatl_ClyTile	-3992.399179
245	LandContour_Bnk	-5001.996997
62	Neighborhood_Gilbert	-5197.585536
59	Neighborhood_CollgCr	-5467.463086
61	Neighborhood_Edwards	-5796.508529

263 rows × 2 columns

So according to this model:

- As OverallQual feature gets bigger the housing price will get bigger.
- Presence of Neighborhood_Edwards will result in smaller median house value.

```
In [75]: 1 x_train_enc['Neighborhood_Edwards']
```

```
Out[75]: 302      0.0
    767      0.0
    429      0.0
   1139      0.0
    558      0.0
    ...
   1041      0.0
   1122      1.0
   1346      0.0
   1406      0.0
   1389      0.0
Name: Neighborhood_Edwards, Length: 1314, dtype: float64
```

Regression score functions

- We aren't doing classification anymore, so we can't just check for equality:

```
In [76]: 1 # This doesn't make sense:
2 lr_tuned.predict(x_train) == y_train
```

```
Out[76]: 302      False
    767      False
    429      False
   1139      False
    558      False
    ...
   1041      False
   1122      False
   1346      False
   1406      False
   1389      False
Name: SalePrice, Length: 1314, dtype: bool
```

```
In [77]: 1 y_train.values
```

```
Out[77]: array([205000, 160000, 175000, ..., 262500, 133000, 131000])
```

```
In [78]: 1 lr_tuned.predict(x_train)
```

```
Out[78]: array([212894.62756285, 178502.78223444, 189937.18327372, ...,
   245233.6751565 , 129863.13373552, 135439.89186716])
```

Processing math: 100% We need a score that reflects how right/wrong each prediction is.

There are a number of popular scoring functions for regression. We are going to look at some common metrics:

- mean squared error (MSE)
- R^2
- root mean squared error (RMSE)
- MAPE

See [sklearn documentation \(\[https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics\]\(https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics\)\)](https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics) for more details.

Mean squared error (MSE)

- A common metric is mean squared error:

```
In [79]: 1 preds = lr_tuned.predict(X_train)
```

```
In [80]: 1 np.mean((y_train - preds) ** 2)
```

Out[80]: 873230473.3636098

Perfect predictions would have MSE=0.

```
In [81]: 1 np.mean((y_train - y_train) ** 2)
```

Out[81]: 0.0

This is also implemented in sklearn:

```
In [82]: 1 from sklearn.metrics import mean_squared_error
2
3 mean_squared_error(y_train, preds)
```

Out[82]: 873230473.3636098

- MSE looks huge and unreasonable. There is an error of $\sim \$1$ Billion!
- Is this score good or bad?

- Unlike classification, with regression **our target has units**.
- The target is in dollars, the mean squared error is in $\$dollars^2$
- The score also depends on the scale of the targets.
- If we were working in cents instead of dollars, our MSE would be 10,000 times (100^2) higher!

```
In [83]: 1 np.mean((y_train * 100 - preds * 100) ** 2)
```

Out[83]: 8732304733636.098

Root mean squared error or RMSE

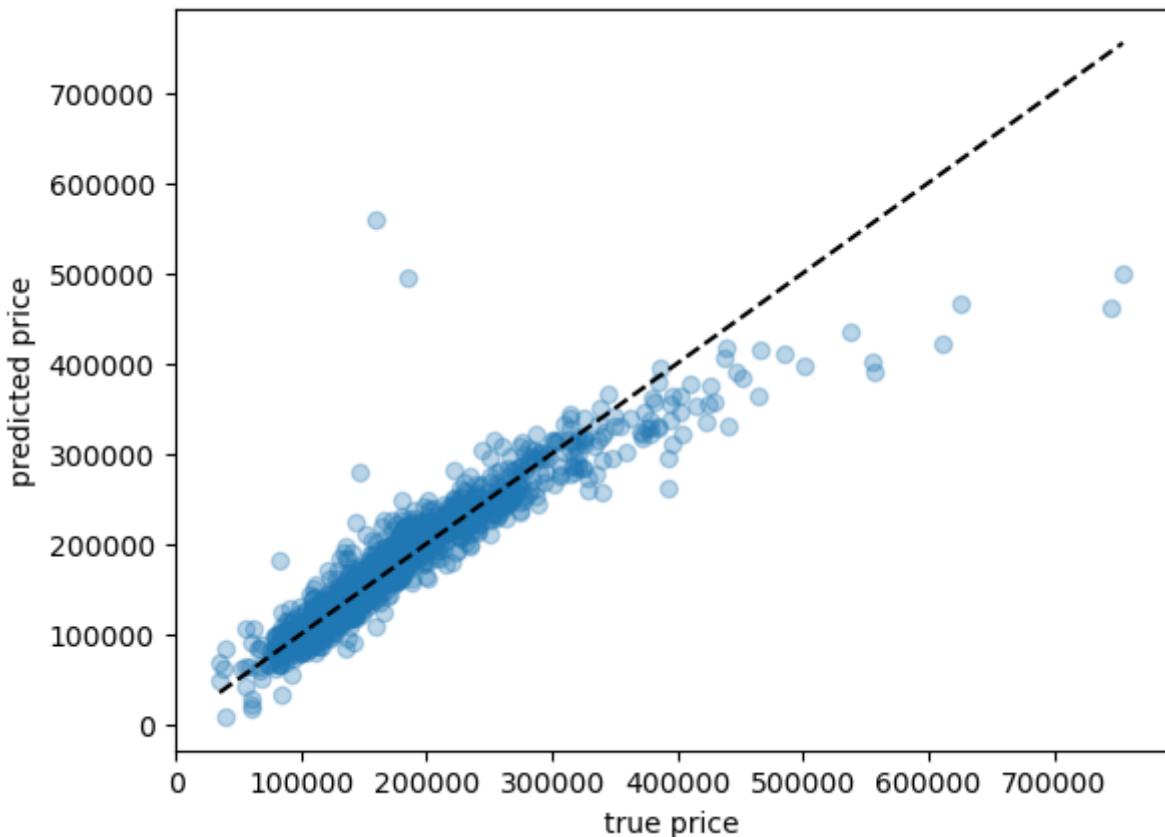
- The MSE above is in \$dollars^2\$.
- A more relatable metric would be the root mean squared error, or RMSE

```
In [84]: 1 np.sqrt(mean_squared_error(y_train, lr_tuned.predict(X_train)))
```

Out[84]: 29550.473318774606

- Error of \$30,000 makes more sense.
- Can we dig deeper?

```
In [85]: 1 plt.scatter(y_train, lr_tuned.predict(X_train), alpha=0.3)
2 grid = np.linspace(y_train.min(), y_train.max(), 1000)
3 plt.plot(grid, grid, "--k")
4 plt.xlabel("true price")
5 plt.ylabel("predicted price");
```



- Here we can see a few cases where our prediction is way off.
- Is there something weird about those houses, perhaps? Outliers?
- Under the line means we're under-predicting, over the line means we're over-predicting.

Processing math: 100%

\$R^2\$ (not in detail)

A common score is the \$R^2\$

- This is the score that `sklearn` uses by default when you call `score()`:
- You can [read about it \(\[https://en.wikipedia.org/wiki/Coefficient_of_determination\]\(https://en.wikipedia.org/wiki/Coefficient_of_determination\)\)](https://en.wikipedia.org/wiki/Coefficient_of_determination) if interested.
- Intuition: similar to mean squared error, but flipped (higher is better), and normalized so the max is 1.

$$\text{R}^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

Key points:

- The maximum is 1 for perfect predictions
- Negative values are very bad: "worse than DummyRegressor" (very bad)

(optional) Warning: MSE is "reversible" but \$R^2\$ is not:

```
In [86]: 1 mean_squared_error(y_train, preds)
```

```
Out[86]: 873230473.3636098
```

```
In [87]: 1 mean_squared_error(preds, y_train)
```

```
Out[87]: 873230473.3636098
```

```
In [88]: 1 r2_score(y_train, preds)
```

```
Out[88]: 0.8601212294857903
```

```
In [89]: 1 r2_score(preds, y_train)
```

```
Out[89]: 0.827962225882707
```

- When you call `fit` it minimizes MSE / maximizes \$R^2\$ (or something like that) by default.
- Just like in classification, this isn't always what you want!!

MAPE

- We got an RMSE of ~\$30,000 before.

Question: Is an error of \$30,000 acceptable?

Processing math: 100%

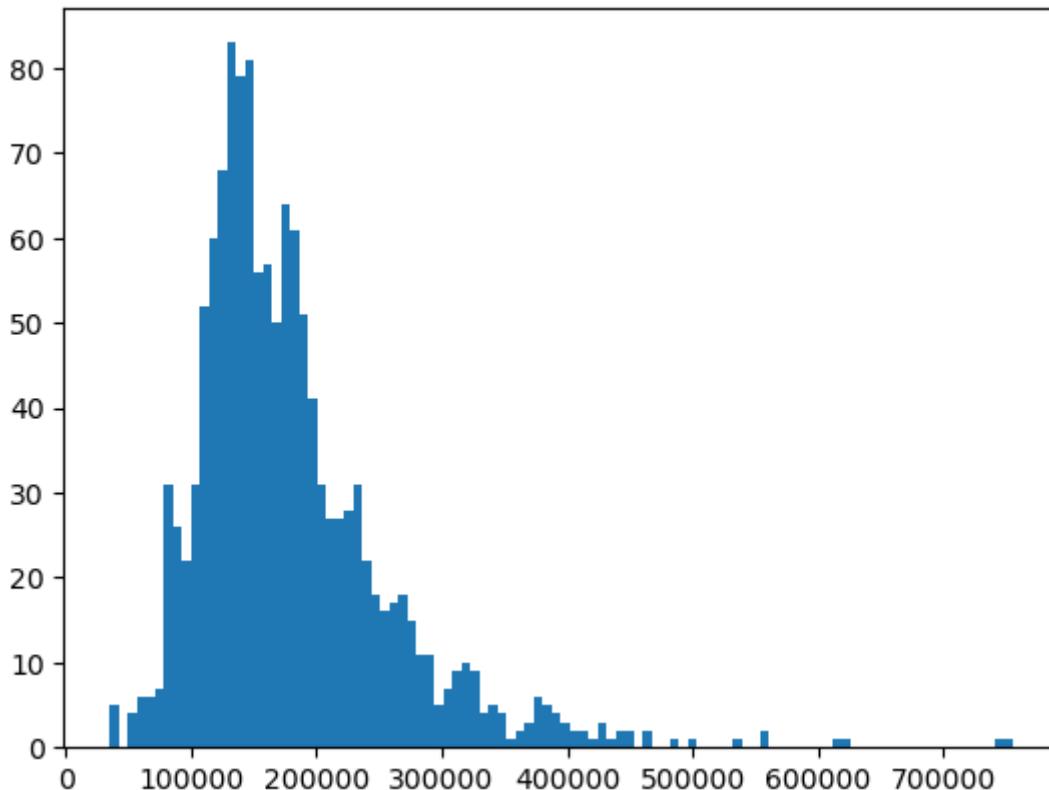
```
In [90]: 1 np.sqrt(mean_squared_error(y_train, lr_tuned.predict(X_train)))
```

```
Out[90]: 29550.473318774606
```

- For a house worth \$600k, it seems reasonable! That's 5% error.
- For a house worth \$60k, that is terrible. It's 50% error.

We have both of these cases in our dataset.

```
In [91]: 1 plt.hist(y_train, bins=100);
```



How about looking at percent error?

Processing math: 100%

```
In [92]: 1 pred_train = lr_tuned.predict(X_train)
          2 percent_errors = (pred_train - y_train) / y_train * 100.0
          3 percent_errors
```

```
Out[92]: 302      3.851038
          767      11.564239
          429      8.535533
          1139     -16.371069
          558      17.177968
          ...
          1041     -0.496571
          1122     -28.696351
          1346     -6.577648
          1406     -2.358546
          1389     3.389230
Name: SalePrice, Length: 1314, dtype: float64
```

These are both positive (predict too high) and negative (predict too low).

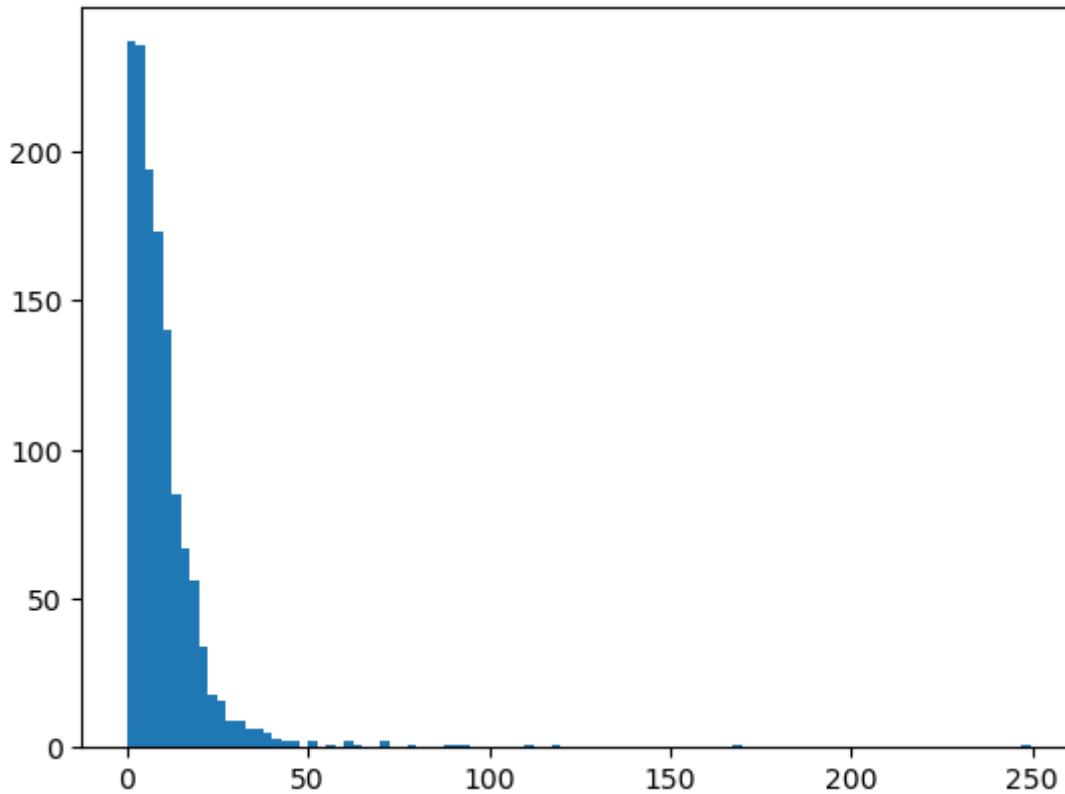
We can look at the absolute percent error:

```
In [93]: 1 np.abs(percent_errors)
```

```
Out[93]: 302      3.851038
          767      11.564239
          429      8.535533
          1139     16.371069
          558      17.177968
          ...
          1041     0.496571
          1122     28.696351
          1346     6.577648
          1406     2.358546
          1389     3.389230
Name: SalePrice, Length: 1314, dtype: float64
```

In [94]:

```
1 plt.hist(np.abs(percent_errors), bins=100);
```



And, like MSE, we can take the average over examples. This is called mean absolute percent error (MAPE).

In [95]:

```
1 def mape(true, pred):
2     return 100.0 * np.mean(np.abs((pred - true) / true))
```

In [96]:

```
1 mape(y_train, pred_train)
```

Out[96]:

```
10.093121294225265
```

- Ok, this is quite interpretable.
- On average, we have around 10% error.

Transforming the targets

- Does `.fit()` know we care about MAPE?
- No, it doesn't. Why are we minimizing MSE (or something similar) if we care about MAPE??
- When minimizing MSE, the expensive houses will dominate because they have the biggest error.
- Which is better for RMSE?

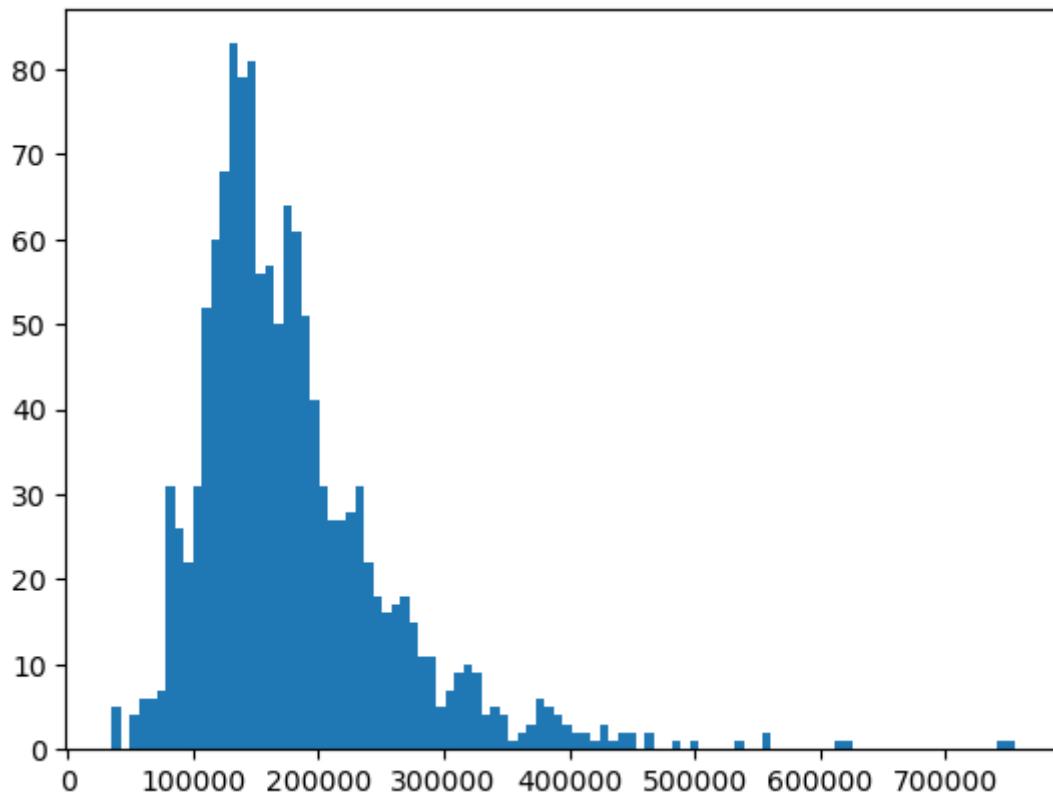
- Example 1: Truth: \$50k, Prediction: \\$100k
- Example 2: Truth: \$500k, Prediction: \\$550k
- RMSE: \$50k
- MAPE: 45%

Model B

- Example 1: Truth: \$50k, Prediction: \\$60k
- Example 2: Truth: \$500k, Prediction: \\$600k
- RMSE: \$71k
- MAPE: 20%

- How can we get `.fit()` to think about MAPE?
- A common practice which tends to work is log transforming the targets.
- That is, transform $y \rightarrow \log(y)$.

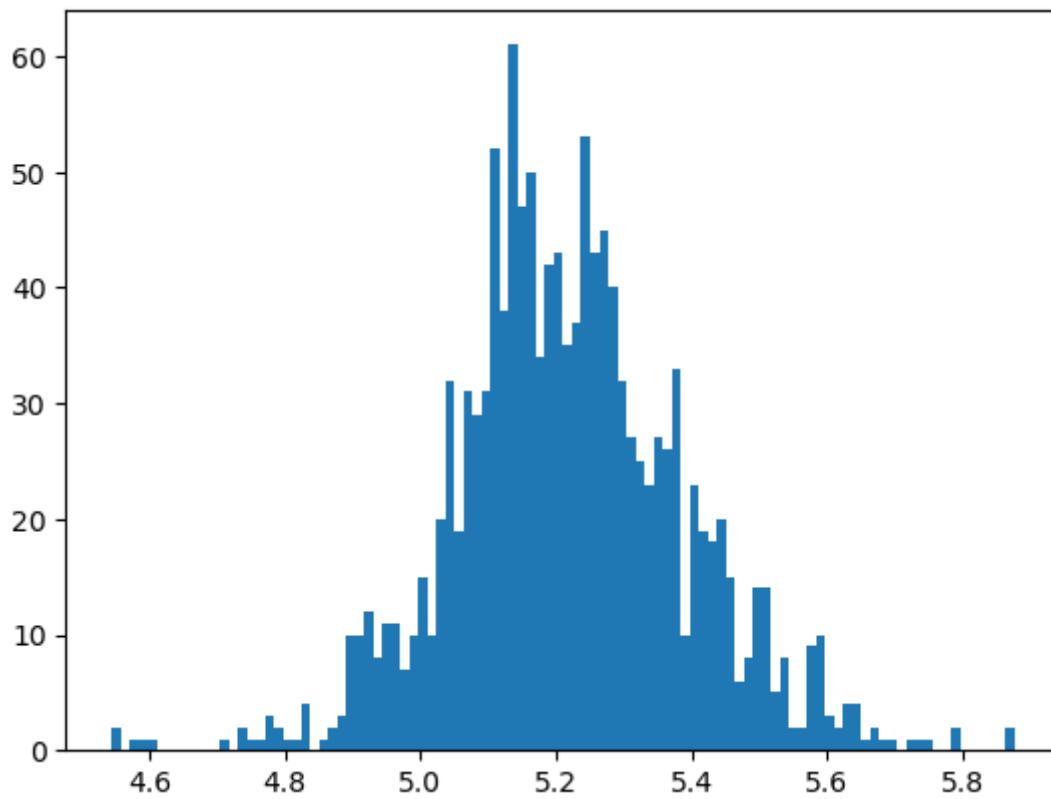
```
In [97]: 1 plt.hist(y_train, bins=100);
```



Processing math: 100%

In [98]:

```
1 plt.hist(np.log10(y_train), bins=100);
```



We can incorporate this in our pipeline using `sklearn`.

In [99]:

```
1 from sklearn.compose import TransformedTargetRegressor
```

In [100]:

```
1 ttr = TransformedTargetRegressor(
2     Ridge(alpha=best_alpha), func=np.log1p, inverse_func=np.expm1
3 ) # transformer for log transforming the target
4 ttr_pipe = make_pipeline(preprocessor, ttr)
```

In [101]:

```
1 ttr_pipe.fit(X_train, y_train); # y_train automatically transformed
```

In [102]:

```
1 ttr_pipe.predict(X_train) # predictions automatically un-transformed
```

Out[102]:

```
array([221355.29528077, 170663.43286226, 182608.09768702, ...,
       248575.94877669, 132148.9047652 , 133262.17638244])
```

In [103]:

```
1 mape(y_test, ttr_pipe.predict(X_test))
```

Out[103]:

```
7.808600924240852
```

We reduced MAPE from ~10% to ~8% with this trick!

Processing math: 100%

Different scoring functions with `cross_validate`

- Let's try using MSE instead of the default R^2 score.

In [104]:

```

1 pd.DataFrame(
2     cross_validate(
3         lr_tuned,
4         X_train,
5         y_train,
6         return_train_score=True,
7         scoring="neg_mean_squared_error",
8     )
9 )

```

Out[104]:

	fit_time	score_time	test_score	train_score
0	0.070120	0.027592	-7.060346e+08	-9.383069e+08
1	0.073525	0.027878	-1.239851e+09	-8.267971e+08
2	0.071608	0.048744	-1.125125e+09	-8.763019e+08
3	0.111845	0.030674	-9.819320e+08	-8.847908e+08
4	0.072874	0.026326	-2.268434e+09	-7.397199e+08

In [105]:

```

1 def mape(true, pred):
2     return 100.0 * np.mean(np.abs((pred - true) / true))
3
4
5 # make a scorer function that we can pass into cross-validation
6 mape_scorer = make_scorer(mape, greater_is_better=True)
7
8 pd.DataFrame(
9     cross_validate(
10         lr_tuned, X_train, y_train, return_train_score=True, scoring=ma
11     )
12 )

```

Out[105]:

	fit_time	score_time	test_score	train_score
0	0.077079	0.025021	9.699277	10.407124
1	0.076896	0.025295	10.803043	9.966190
2	0.094964	0.030100	11.836195	10.180734
3	0.090991	0.037569	10.784686	10.247198
4	0.081499	0.029051	12.196718	9.828607

Processing math: 100%

In [106]:

```

1 scoring = {
2     "r2": "r2",
3     "mape_scorer": mape_scorer,
4     "neg_root_mean_square_error": "neg_root_mean_squared_error",
5     "neg_mean_squared_error": "neg_mean_squared_error",
6 }
7
8 pd.DataFrame(
9     cross_validate(lr_tuned, X_train, y_train, return_train_score=True,
10 ).T

```

Out[106]:

	0	1	2	3
fit_time	8.490086e-02	8.591008e-02	8.298492e-02	8.994293e-02
score_time	3.135419e-02	3.481293e-02	3.112602e-02	3.396201e-02
test_r2	8.668969e-01	8.200460e-01	8.262644e-01	8.511854e-01
train_r2	8.551369e-01	8.636241e-01	8.579735e-01	8.561893e-01
test_mape_scorer	9.699277e+00	1.080304e+01	1.183620e+01	1.078469e+01
train_mape_scorer	1.040712e+01	9.966190e+00	1.018073e+01	1.024720e+01
test_neg_root_mean_square_error	-2.657131e+04	-3.521152e+04	-3.354288e+04	-3.133579e+04
train_neg_root_mean_square_error	-3.063179e+04	-2.875408e+04	-2.960240e+04	-2.974543e+04
test_neg_mean_squared_error	-7.060346e+08	-1.239851e+09	-1.125125e+09	-9.819320e+08
train_neg_mean_squared_error	-9.383069e+08	-8.267971e+08	-8.763019e+08	-8.847908e+08

In [107]:

```
1 mape(y_test, lr_tuned.predict(X_test))
```

Out[107]:

9.496387589496008

Using regression metrics with scikit-learn

- In sklearn you will notice that it has negative version of the metrics above (e.g., neg_mean_squared_error, neg_root_mean_squared_error).
- The reason for this is that scores return a value to maximize, the higher the better.
- If you define your own scorer function and if you do not want this interpretation, you can set the greater_is_better parameter to False

Questions for class discussion

True/False

1. Price per square foot would be a good feature to add in our X .
2. The alpha hyperparameter of Ridge has similar interpretation of C hyperparameter of LogisticRegression ; higher alpha means more complex model.

Processing math: 100%

3. In regression, one should use MAPE instead of MSE when relative (percent) error matters more than absolute error.
4. A lower RMSE value indicates a better model.
5. We can still use precision and recall for regression problems but now we have other

Summary

- House prices dataset target is price, which is numeric -> regression rather than classification
- There are corresponding versions of all the tools we used:
 - `DummyClassifier` -> `DummyRegressor`
 - `LogisticRegression` -> `Ridge`
- Ridge hyperparameter `alpha` is like `LogisticRegression` hyperparameter `c`, but opposite meaning
- We'll avoid `LinearRegression` in this course.

- Scoring metrics
- `R^2` is the default `.score()`, it is unitless, 0 is bad, 1 is best
- MSE (mean squared error) is in units of target squared, hard to interpret; 0 is best
- RMSE (root mean squared error) is in the same units as the target; 0 is best
- MAPE (mean average percent error) is unitless; 0 is best, 100 is bad

In []:

1

Processing math: 100%

Lecture 11: Ensembles

UBC 2022-23

Instructor: Mathias Lécuyer

The interests of truth require a diversity of opinions.
by John Stuart Mill

Imports

In [1]:

```
1 import os
2
3 %matplotlib inline
4 import string
5 import sys
6 from collections import deque
7
8 import matplotlib.pyplot as plt
9 import numpy as np
10 import pandas as pd
11
12 sys.path.append("../code/.")
13
14 from plotting_functions import *
15 from sklearn import datasets
16 from sklearn.compose import ColumnTransformer, make_column_transformer
17 from sklearn.dummy import DummyClassifier, DummyRegressor
18 from sklearn.ensemble import RandomForestClassifier, RandomForestRegres
19 from sklearn.impute import SimpleImputer
20 from sklearn.linear_model import LogisticRegression
21 from sklearn.model_selection import (
22     GridSearchCV,
23     RandomizedSearchCV,
24     cross_val_score,
25     cross_validate,
26     train_test_split,
27 )
28 from sklearn.pipeline import Pipeline, make_pipeline
29 from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, Standar
30 from sklearn.svm import SVC, SVR
31 from sklearn.tree import DecisionTreeClassifier
32 from utils import *
```

Lecture learning objectives

From this lecture, you will be able to

- Use `scikit-learn`'s `RandomForestClassifier` and explain its main hyperparameters.
- Explain randomness in the random forest algorithm.
- Use other tree-based models such as as `XGBoost` and `LGBM`.
- Employ ensemble classifier approaches, in particular model averaging and stacking.
- Explain voting and stacking and the differences between them.
- Use `scikit-learn` implementations of these ensemble methods.

Motivation

- **Ensembles** are models that combine multiple machine learning models to create more powerful models.

The Netflix prize

[Source \(<https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429>\)](https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429)

- Most of the winning solutions for Kaggle competitions involve some kind of ensembling. For example:

Key idea: Groups can often make better decisions than individuals, especially when group members are diverse enough.

[The Wisdom of Crowds \(<http://wisdomofcrowds.blogspot.com/2009/12/introduction-part-i.html>\)](http://wisdomofcrowds.blogspot.com/2009/12/introduction-part-i.html)

Tree-based ensemble models

- A number of ensemble models in ML literature.
- Most successful ones on a variety of datasets are tree-based models.
- We'll briefly talk about two such models:
 - Random forests
 - Gradient boosted trees
- We'll also talk about averaging and stacking.

Tree-based models

- Decision tree models are
 - Interpretable
 - They can capture non-linear relationships
 - They don't require scaling of the data and theoretically can work with categorical features.
- But a single decision tree is likely to overfit.

- Key idea: Combine multiple trees to build stronger models.
- These kinds of models are extremely popular in industry and machine learning competitions

Data

- Let's work with [the adult census data set \(<https://www.kaggle.com/uciml/adult-census-income>\)](https://www.kaggle.com/uciml/adult-census-income).

```
In [2]: 1 adult_df_large = pd.read_csv("../data/adult.csv")
2 train_df, test_df = train_test_split(adult_df_large, test_size=0.2, ran
3 train_df_nan = train_df.replace("?", np.NaN)
4 test_df_nan = test_df.replace("?", np.NaN)
5 train_df_nan.head()
```

Out[2]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship
5514	26	Private	256263	HS-grad		9	Never-married	Craft-repair
19777	24	Private	170277	HS-grad		9	Never-married	Other-service
10781	36	Private	75826	Bachelors		13	Divorced	Adm-clerical
32240	22	State-gov	24395	Some-college		10	Married-civ-spouse	Adm-clerical
9876	31	Local-gov	356689	Bachelors		13	Married-civ-spouse	Prof-specialty

```
In [3]: 1 numeric_features = ["age", "fnlwgt", "capital.gain", "capital.loss", "h  
2 categorical_features = [  
3     "workclass",  
4     "marital.status",  
5     "occupation",  
6     "relationship",  
7     "native.country",  
8 ]  
9 ordinal_features = ["education"]  
10 binary_features = ["sex"]  
11 drop_features = ["race", "education.num"]  
12 target_column = "income"
```

```
In [4]: 1 education_levels = [  
2     "Preschool",  
3     "1st-4th",  
4     "5th-6th",  
5     "7th-8th",  
6     "9th",  
7     "10th",  
8     "11th",  
9     "12th",  
10    "HS-grad",  
11    "Prof-school",  
12    "Assoc-voc",  
13    "Assoc-acdm",  
14    "Some-college",  
15    "Bachelors",  
16    "Masters",  
17    "Doctorate",  
18 ]
```

```
In [5]: 1 assert set(education_levels) == set(train_df["education"].unique())
```

In [6]:

```

1 numeric_transformer = make_pipeline(StandardScaler())
2
3 ordinal_transformer = make_pipeline(
4     OrdinalEncoder(categories=[education_levels], dtype=int)
5 )
6
7 categorical_transformer = make_pipeline(
8     SimpleImputer(strategy="constant", fill_value="missing"),
9     OneHotEncoder(handle_unknown="ignore", sparse=False),
10 )
11
12 binary_transformer = make_pipeline(
13     SimpleImputer(strategy="constant", fill_value="missing"),
14     OneHotEncoder(drop="if_binary", dtype=int),
15 )
16
17 preprocessor = make_column_transformer(
18     (numeric_transformer, numeric_features),
19     (ordinal_transformer, ordinal_features),
20     (binary_transformer, binary_features),
21     (categorical_transformer, categorical_features),
22     ("drop", drop_features),
23 )

```

In [7]:

```

1 X_train = train_df_nan.drop(columns=[target_column])
2 y_train = train_df_nan[target_column]
3
4 X_test = test_df_nan.drop(columns=[target_column])
5 y_test = test_df_nan[target_column]

```

Do we have class imbalance?

- There is class imbalance. But without any context, both classes seem equally important.
- Let's use accuracy as our metric.

In [8]:

```
1 train_df_nan["income"].value_counts(normalize=True)
```

Out[8]:

<=50K	0.757985
>50K	0.242015
Name:	income, dtype: float64

In [9]:

```
1 scoring_metric = "accuracy"
```

Let's store all the results in a dictionary called `results`.

In [10]:

```
1 results = {}
```

Baselines

DummyClassifier baseline

```
In [11]: 1 dummy = DummyClassifier(strategy="stratified")
2 results[ "Dummy" ] = mean_std_cross_val_scores(
3     dummy, X_train, y_train, return_train_score=True, scoring=scoring_m
4 )
```

DecisionTreeClassifier baseline

- Let's try decision tree classifier on our data.

```
In [12]: 1 pipe_dt = make_pipeline(preprocessor, DecisionTreeClassifier(random_st
2 results[ "Decision tree" ] = mean_std_cross_val_scores(
3     pipe_dt, X_train, y_train, return_train_score=True, scoring=scoring_m
4 )
5 pd.DataFrame(results).T
```

Out[12]:

	fit_time	score_time	test_score	train_score
Decision tree	0.152 (+/- 0.006)	0.015 (+/- 0.001)	0.813 (+/- 0.003)	1.000 (+/- 0.000)
Dummy	0.008 (+/- 0.001)	0.005 (+/- 0.000)	0.632 (+/- 0.004)	0.633 (+/- 0.003)

Decision tree is clearly overfitting.

Random forests**General idea**

- A single decision tree is likely to overfit
- Random forests use a collection of diverse decision trees
- Each tree overfits on some part of the data but we can reduce overfitting by averaging the results
 - can be shown mathematically

RandomForestClassifier

- Before understanding the details let's first try it out.

In [13]:

```

1 from sklearn.ensemble import RandomForestClassifier
2
3 pipe_rf = make_pipeline(
4     preprocessor, RandomForestClassifier(random_state=123, n_jobs=-1)
5 )
6 results[ "Random forests" ] = mean_std_cross_val_scores(
7     pipe_rf, X_train, y_train, return_train_score=True, scoring=scoring
8 )
9 pd.DataFrame(results).T

```

Out[13]:

	fit_time	score_time	test_score	train_score
Decision tree	0.008 (+/- 0.001)	0.005 (+/- 0.000)	0.632 (+/- 0.004)	0.633 (+/- 0.003)
Random forests	0.152 (+/- 0.006)	0.015 (+/- 0.001)	0.813 (+/- 0.003)	1.000 (+/- 0.000)
Dummy	1.474 (+/- 2.339)	0.041 (+/- 0.004)	0.857 (+/- 0.004)	1.000 (+/- 0.000)

The validation scores are better although it seems like we are still overfitting.

How do they work?

- Decide how many decision trees we want to build
 - can control with `n_estimators` hyperparameter
- fit a diverse set of that many decision trees by **injecting randomness** in the classifier construction
- predict by voting (classification) or averaging (regression) of predictions given by individual models

Inject randomness in the classifier construction

To ensure that the trees in the random forest are different we inject randomness in two ways:

1. Data: **Build each tree on a bootstrap sample** (i.e., a sample drawn **with replacement** from the training set)
2. Features: **At each node, select a random subset of features** (controlled by `max_features` in `scikit-learn`) and look for the best possible split involving one of these features

An example of a bootstrap sample Suppose this is your original dataset: [1,2,3,4]

- a sample drawn with replacement: [1,1,3,4]
- a sample drawn with replacement: [3,2,2,2]
- a sample drawn with replacement: [1,2,4,4]
- ...

There is also something called [ExtraTreesClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html) (<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html>), where we add more randomness by considering a random subset of features at each split and a the **best threshold among a set of random thresholds**.

The random forests classifier

- Create a collection (ensemble) of trees. Grow each tree on an independent bootstrap sample from the data.
- At each node:
 - Randomly select a subset of features out of all features (independently for each node).
 - Find the best split on the selected features.
 - Grow the trees to maximum depth.
- At prediction time:
 - Vote the trees to get predictions for new example.

Example

- Let's create a random forest with 3 estimators.
- Using `max_depth=2` for easy visualization.

In [14]:

```
1 pipe_rf_demo = make_pipeline(  
2     preprocessor, RandomForestClassifier(max_depth=2, n_estimators=3, r  
3 )  
4 pipe_rf_demo.fit(X_train, y_train);
```

- Let's get the feature names of transformed features.

In [15]:

```
1 feature_names = (
2     numeric_features
3     + ordinal_features
4     + binary_features
5     + list(
6         pipe_rf_demo.named_steps["columntransformer"]
7         .named_transformers_[ "pipeline-4" ]
8         .named_steps[ "onehotencoder" ]
9         .get_feature_names_out()
10    )
11 )
12 feature_names[:10]
```

Out[15]:

```
['age',
 'fnlwgt',
 'capital.gain',
 'capital.loss',
 'hours.per.week',
 'education',
 'sex',
 'x0_Federal-gov',
 'x0_Local-gov',
 'x0_Never-worked']
```

- Let's sample a test example.

In [16]:

```
1 test_example = X_test.sample(1)
2 print("Classes: ", pipe_rf_demo.classes_)
3 print("Prediction by random forest: ", pipe_rf_demo.predict(test_example))
4 transformed_example = preprocessor.transform(test_example)
5 pd.DataFrame(data=transformed_example.flatten(), index=feature_names)
```

Classes: ['<=50K' '>50K']
Prediction by random forest: ['<=50K']

Out[16]:

	0
age	-0.333171
fnlwgt	-0.924948
capital.gain	0.499907
capital.loss	-0.217680
hours.per.week	-0.042081
...	...
x4_Trinadad&Tobago	0.000000
x4_United-States	1.000000
x4_Vietnam	0.000000
x4_Yugoslavia	0.000000
x4_missing	0.000000

86 rows × 1 columns

- We can look at different trees created by the random forest.
- Note that each tree looks at different set of features and slightly different datasets.

In [55]:

```
1 for i, tree in enumerate(
2     pipe_rf_demo.named_steps["randomforestclassifier"].estimators_
3 ):  
4     print("\n\nTree", i + 1)
5     display(display_tree(feature_names, tree))
6     print("prediction", tree.predict(preprocessor.transform(test_exampl
```

Tree 1

```
<graphviz.sources.Source at 0x199c5bb80>
prediction [0.]
```

Tree 2

```
<graphviz.sources.Source at 0x199c5b5b0>
prediction [0.]
```

Tree 3

```
<graphviz.sources.Source at 0x199efc730>
prediction [0.]
```

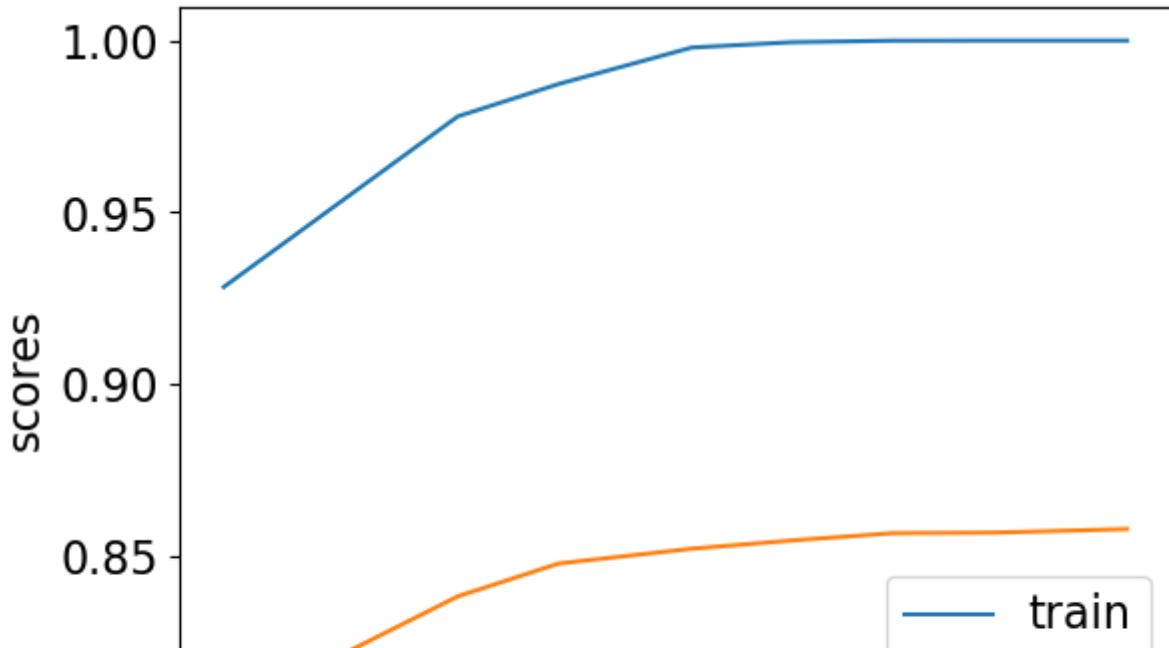
Some important hyperparameters:

- `n_estimators` : number of decision trees (higher = more complexity)
- `max_depth` : max depth of each decision tree (higher = more complexity)
- `max_features` : the number of features you get to look at each split (higher = more complexity; lower = more randomness = more diversity across estimators)

Random forests: number of trees (`n_estimators`) and the fundamental tradeoff

In [56]:

```
1 make_num_tree_plot(
2     preprocessor, X_train, y_train, X_test, y_test, [1, 5, 10, 25, 50,
3 )
```



Number of trees and fundamental trade-off

- Above: seems like we're beating the fundamental "tradeoff" by increasing training score and not decreasing validation score much.
- This is the promise of ensembles, though it's not guaranteed to work so nicely.

Here, more trees are always better! We pick less trees for speed.

Strengths and weaknesses

- Usually one of the best performing off-the-shelf classifiers without heavy tuning of hyperparameters,
- Doesn't require scaling of data,
- Less likely to overfit,
- Slower than decision trees because we are fitting multiple trees but can easily parallelize training because all trees are independent of each other,
- In general, able to capture a much broader picture of the data compared to a single decision tree.

Weaknesses

- Require more memory,

- Hard to interpret,
- Tend not to perform well on high dimensional sparse data such as text data.

Make sure to set the `random_state` for reproducibility. Changing the `random_state` can have a big impact on the model and the results due to the random nature of these models. Having more trees can get you a more robust estimate.

[The original random forests paper \(<https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf>\)](https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf), by Leo Breiman.

Gradient boosted trees

Another popular and effective class of tree-based ensemble models is gradient boosted trees.

- No randomization.
- The key idea is to combine many simple models called weak learners to create a strong learner.
- They combine multiple shallow (depth 1 to 5) decision trees
- They build trees in a serial manner, where each tree tries to correct the mistakes of previous ones.

Important hyperparameters

- `n_estimators`
 - control the number of trees to build
- `learning_rate`
 - controls how strongly each tree tries to correct the mistakes of the previous trees
 - higher learning rate means each tree can make stronger corrections, which means more complex model

We'll not go into the details. We'll look at brief examples of using the following three gradient boosted tree models.

- [XGBoost \(<https://xgboost.readthedocs.io/en/latest/>\)](https://xgboost.readthedocs.io/en/latest/)
- [LightGBM \(<https://lightgbm.readthedocs.io/en/latest/Python-Intro.html>\)](https://lightgbm.readthedocs.io/en/latest/Python-Intro.html)
- [CatBoost \(<https://catboost.ai/docs/concepts/python-quickstart.html>\)](https://catboost.ai/docs/concepts/python-quickstart.html)

[XGBoost \(<https://xgboost.ai/about>\)](https://xgboost.ai/about)

- Not part of `sklearn` but has similar interface.
- Install it in your conda environment: `conda install -c conda-forge xgboost`
- Supports missing values
- GPU training, networked parallel training
- Supports sparse data
- Typically better scores than random forests

[LightGBM \(<https://lightgbm.readthedocs.io/>\)](https://lightgbm.readthedocs.io/)

- Not part of `sklearn` but has similar interface.
- Install it in your conda environment: `conda install -c conda-forge lightgbm`
- Small model size
- Faster
- Typically better scores than random forests

[CatBoost \(<https://catboost.ai/>\)](https://catboost.ai/)

- Not part of `sklearn` but has similar interface.
- Install it in your conda environment: `conda install -c conda-forge catboost`
- Usually better scores but slower compared to XGBoost and LightGBM

In [57]:

```

1 from catboost import CatBoostClassifier
2 from lightgbm.sklearn import LGBMClassifier
3 from sklearn.tree import DecisionTreeClassifier
4 from xgboost import XGBClassifier
5
6 from sklearn.compose import TransformedTargetRegressor
7 from sklearn.preprocessing import LabelEncoder
8
9 pipe_lr = make_pipeline(
10     preprocessor, LogisticRegression(max_iter=2000, random_state=123)
11 )
12 pipe_dt = make_pipeline(preprocessor, DecisionTreeClassifier(random_st
13 pipe_rf = make_pipeline(preprocessor, RandomForestClassifier(random_st
14 pipe_xgb = make_pipeline(
15     preprocessor, XGBClassifier(random_state=123, eval_metric="logloss"
16 )
17 pipe_lgbm = make_pipeline(preprocessor, LGBMClassifier(random_state=123
18 pipe_catboost = make_pipeline(
19     preprocessor, CatBoostClassifier(verbose=0, random_state=123)
20 )
21 # XGBoost requires numeric targets
22 label_encoder = LabelEncoder()
23 label_encoder.fit(y_train)
24
25 classifiers = {
26     "logistic regression": pipe_lr,
27     "decision tree": pipe_dt,
28     "random forest": pipe_rf,
29     "XGBoost": pipe_xgb,
30     "LightGBM": pipe_lgbm,
31     "CatBoost": pipe_catboost,
32 }
```

In [58]:

```

1 import warnings
2
3 warnings.simplefilter(action="ignore", category=DeprecationWarning)
4 warnings.simplefilter(action="ignore", category=UserWarning)
```

In [59]:

```
1 results = {}
```

In [60]:

```

1 dummy = DummyClassifier(strategy="stratified")
2 results["Dummy"] = mean_std_cross_val_scores(
3     dummy, X_train, y_train, return_train_score=True, scoring=scoring_m
4 )
```

In [61]:

```

1 for (name, model) in classifiers.items():
2     results[name] = mean_std_cross_val_scores(
3         model, X_train, label_encoder.transform(y_train), return_train_
4     )
```

In [62]: 1 pd.DataFrame(results).T

Out[62]:

		fit_time	score_time	test_score	train_score
	Dummy	0.007 (+/- 0.000)	0.006 (+/- 0.000)	0.631 (+/- 0.006)	0.634 (+/- 0.003)
	logistic regression	1.316 (+/- 0.044)	0.010 (+/- 0.000)	0.850 (+/- 0.006)	0.851 (+/- 0.001)
	decision tree	0.144 (+/- 0.002)	0.010 (+/- 0.001)	0.813 (+/- 0.003)	1.000 (+/- 0.000)
	random forest	1.398 (+/- 0.028)	0.082 (+/- 0.002)	0.857 (+/- 0.004)	1.000 (+/- 0.000)
	XGBoost	1.066 (+/- 0.031)	0.015 (+/- 0.000)	0.870 (+/- 0.003)	0.909 (+/- 0.002)
	LightGBM	0.740 (+/- 0.062)	0.016 (+/- 0.001)	0.871 (+/- 0.004)	0.892 (+/- 0.000)
	CatBoost	4.566 (+/- 0.075)	0.079 (+/- 0.001)	0.872 (+/- 0.003)	0.900 (+/- 0.001)

Some observations

- Keep in mind all these results are with default hyperparameters
- Ideally we would carry out hyperparameter optimization for all of them and then compare the results.
- We are using a particular scoring metric (accuracy in this case)
- We are scaling numeric features but it shouldn't matter for these tree-based models.
- Look at the std. Doesn't look very high.
 - The scores look more or less stable.

In [63]: 1 pd.DataFrame(results).T

Out[63]:

		fit_time	score_time	test_score	train_score
	Dummy	0.007 (+/- 0.000)	0.006 (+/- 0.000)	0.631 (+/- 0.006)	0.634 (+/- 0.003)
	logistic regression	1.316 (+/- 0.044)	0.010 (+/- 0.000)	0.850 (+/- 0.006)	0.851 (+/- 0.001)
	decision tree	0.144 (+/- 0.002)	0.010 (+/- 0.001)	0.813 (+/- 0.003)	1.000 (+/- 0.000)
	random forest	1.398 (+/- 0.028)	0.082 (+/- 0.002)	0.857 (+/- 0.004)	1.000 (+/- 0.000)
	XGBoost	1.066 (+/- 0.031)	0.015 (+/- 0.000)	0.870 (+/- 0.003)	0.909 (+/- 0.002)
	LightGBM	0.740 (+/- 0.062)	0.016 (+/- 0.001)	0.871 (+/- 0.004)	0.892 (+/- 0.000)
	CatBoost	4.566 (+/- 0.075)	0.079 (+/- 0.001)	0.872 (+/- 0.003)	0.900 (+/- 0.001)

- Decision trees and random forests overfit
 - Other models do not seem to overfit much.
- Fit times
 - Decision trees are fast but not very accurate
 - LightGBM is faster than decision trees and more accurate!
 - CatBoost fit time is highest followed by random forests.
 - There is not much difference between the validation scores of XGBoost, LightGBM, and CatBoost but it is about 48x slower than LightGBM!
 - XGBoost and LightGBM are faster and more accurate than random forest!
- Scores times

- Prediction times are much smaller in all cases.

What classifier should I use?

Simple answer

- Whichever gets the highest CV score making sure that you're not overusing the validation set.

Interpretability

- This is an area of growing interest and concern in ML.
- How important is interpretability for you?
- In the next class we'll talk about interpretability of non-linear models.

Speed/code maintenance

- Other considerations could be speed (fit and/or predict), maintainability of the code.

Finally, you could use all of them!

Averaging

Earlier we looked at a bunch of classifiers:

```
In [64]: 1 classifiers.keys()
```

```
Out[64]: dict_keys(['logistic regression', 'decision tree', 'random forest', 'XGBoost', 'LightGBM', 'CatBoost'])
```

What if we use all these models and let them vote during prediction time?

```
In [65]: 1 from sklearn.ensemble import VotingClassifier
2
3 averaging_model = VotingClassifier(
4     list(classifiers.items()), voting="soft"
5 ) # need the list() here for cross_val to work!
```

In [66]:

```
1 from sklearn import set_config  
2  
3 set_config(display="diagram") # global setting
```

In [67]: 1 averaging_model

```
Out[67]: votingClassifier(estimators=[('logistic regression',
                                         Pipeline(steps=[('columntransformer',
                                                               ColumnTransformer(transformers=[('pipeline-1',
                                                                 Pipeline(steps=[('standardscaler',
                                                                 StandardScaler())])),
                                                               ['age',
                                                                'fnlwgt',
                                                                'capital.gain',
                                                                'capital.loss',
                                                                'hours.per.week']),
                                                               ('pipeline-2',
                                                                Pipeline(steps=[('ordinalencoder',
                                                                 OrdinalEncoder(categories=[[ 'Preschool',
                                                                 '1st-4th']...,
                                                                Pipeline(steps=[('simpleimputer',
                                                                 SimpleImputer(fill_value='missing',
                                                                strategy='constant'))),
                                                                ('onehotencoder',
                                                                OneHotEncoder(handle_unknown='ignore',
                                                                sparse=False))]),
                                                               ['workclass',
                                                                'marital.status',
                                                                'occupation',
                                                                'relationship',
                                                                'native.country'])],
```

```
( 'drop',
  'drop',
  [ 'race',
    'education.num' ] ) ) ) ),
( 'catboostclassifier',
<catboost.core.CatBoostCla
ssifier object at 0x19aca82b0>) ) ) ],
      voting='soft')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

This VotingClassifier will take a vote using the predictions of the constituent classifier pipelines.

Main parameter: voting

- voting='hard'
 - it uses the output of predict and actually votes.
- voting='soft'
 - with voting='soft' it averages the output of predict_proba and then thresholds / takes the larger.

- The choice depends on whether you trust predict_proba from your base classifiers - if so, it's nice to access that information.

In [68]: 1 averaging_model.fit(X_train, y_train);

- What happens when you fit a VotingClassifier ?
 - It will fit all constituent models.

It seems sklearn requires us to actually call `fit` on the `VotingClassifier`, instead of passing in pre-fit models. This is an implementation choice rather than a conceptual limitation.

Let's look at particular test examples where income is ">50k" (y=1):

```
In [69]: 1 test_g50k = (
2     test_df.query("income == '>50K'").sample(4, random_state=2).drop(co
3 )
4 test_150k = (
5     test_df.query("income == '<=50K'")
6     .sample(4, random_state=2)
7     .drop(columns=["income"]))
8 )
```

```
In [70]: 1 averaging_model.classes_
```

```
Out[70]: array(['<=50K', '>50K'], dtype=object)
```

```
In [71]: 1 data = {"Voting classifier": averaging_model.predict(test_g50k)}
2 pd.DataFrame(data)
```

```
Out[71]: Voting classifier
```

0	>50K
1	>50K
2	>50K
3	<=50K

For hard voting, these are the votes:

```
In [72]: 1 r1 = {
2     name: classifier.predict(test_g50k)
3     for name, classifier in averaging_model.named_estimators_.items()
4 }
5 data.update(r1)
6 pd.DataFrame(data)
```

	Voting classifier	logistic regression	decision tree	random forest	XGBoost	LightGBM	CatBoost
0	>50K	1	1	1	1	1	1
1	>50K	1	1	1	1	1	1
2	>50K	1	0	1	1	1	1
3	<=50K	0	0	0	0	0	0

For soft voting, these are the scores:

In [73]:

```

1 r1 = {
2     name: classifier.predict_proba(test_g50k)
3     for name, classifier in averaging_model.named_estimators_.items()
4 }
5 r1

```

```

Out[73]: {'logistic regression': array([[2.28705943e-14, 1.00000000e+00],
   [4.18744244e-01, 5.81255756e-01],
   [4.96637268e-01, 5.03362732e-01],
   [8.87444717e-01, 1.12555283e-01]]),
'decision tree': array([[0., 1.],
   [0., 1.],
   [1., 0.],
   [1., 0.]]),
'random forest': array([[0. , 1. ],
   [0.31, 0.69],
   [0.37, 0.63],
   [0.57, 0.43]]),
'XGBoost': array([[0.00168145, 0.99831855],
   [0.30067134, 0.69932866],
   [0.31893963, 0.6810604 ],
   [0.78900903, 0.21099098]], dtype=float32),
'LightGBM': array([[0.00187645, 0.99812355],
   [0.28722892, 0.71277108],
   [0.28457261, 0.71542739],
   [0.8095596 , 0.1904404 ]]),
'CatBoost': array([[0.00136353, 0.99863647],
   [0.26605365, 0.73394635],
   [0.32779854, 0.67220146],
   [0.82750574, 0.17249426]])}

```

(Aside: the probability scores from `DecisionTreeClassifier` are pretty bad)

Let's see how well this model performs.

In [74]:

```
1 results["Voting"] = mean_std_cross_val_scores(averaging_model, X_train,
```

In [75]:

```
1 pd.DataFrame(results).T
```

Out[75]:

	fit_time	score_time	test_score	train_score
Dummy	0.007 (+/- 0.000)	0.006 (+/- 0.000)	0.631 (+/- 0.006)	0.634 (+/- 0.003)
logistic regression	1.316 (+/- 0.044)	0.010 (+/- 0.000)	0.850 (+/- 0.006)	0.851 (+/- 0.001)
decision tree	0.144 (+/- 0.002)	0.010 (+/- 0.001)	0.813 (+/- 0.003)	1.000 (+/- 0.000)
random forest	1.398 (+/- 0.028)	0.082 (+/- 0.002)	0.857 (+/- 0.004)	1.000 (+/- 0.000)
XGBoost	1.066 (+/- 0.031)	0.015 (+/- 0.000)	0.870 (+/- 0.003)	0.909 (+/- 0.002)
LightGBM	0.740 (+/- 0.062)	0.016 (+/- 0.001)	0.871 (+/- 0.004)	0.892 (+/- 0.000)
CatBoost	4.566 (+/- 0.075)	0.079 (+/- 0.001)	0.872 (+/- 0.003)	0.900 (+/- 0.001)
Voting	9.208 (+/- 0.147)	0.222 (+/- 0.003)	0.868 (+/- 0.003)	NaN

It appears that here we didn't do better than our best classifier :(.

Let's try removing decision tree classifier.

```
In [76]: 1 classifiers_ndt = classifiers.copy()
2 del classifiers_ndt["decision tree"]
3 averaging_model_ndt = VotingClassifier(
4     list(classifiers_ndt.items()), voting="soft"
5 ) # need the list() here for cross_val to work!
6
7 results["Voting_ndt"] = mean_std_cross_val_scores(
8     averaging_model_ndt,
9     X_train,
10    y_train,
11    return_train_score=True,
12    scoring=scoring_metric,
13 )
```

```
In [77]: 1 pd.DataFrame(results).T
```

	fit_time	score_time	test_score	train_score
Dummy	0.007 (+/- 0.000)	0.006 (+/- 0.000)	0.631 (+/- 0.006)	0.634 (+/- 0.003)
logistic regression	1.316 (+/- 0.044)	0.010 (+/- 0.000)	0.850 (+/- 0.006)	0.851 (+/- 0.001)
decision tree	0.144 (+/- 0.002)	0.010 (+/- 0.001)	0.813 (+/- 0.003)	1.000 (+/- 0.000)
random forest	1.398 (+/- 0.028)	0.082 (+/- 0.002)	0.857 (+/- 0.004)	1.000 (+/- 0.000)
XGBoost	1.066 (+/- 0.031)	0.015 (+/- 0.000)	0.870 (+/- 0.003)	0.909 (+/- 0.002)
LightGBM	0.740 (+/- 0.062)	0.016 (+/- 0.001)	0.871 (+/- 0.004)	0.892 (+/- 0.000)
CatBoost	4.566 (+/- 0.075)	0.079 (+/- 0.001)	0.872 (+/- 0.003)	0.900 (+/- 0.001)
Voting	9.208 (+/- 0.147)	0.222 (+/- 0.003)	0.868 (+/- 0.003)	NaN
Voting_ndt	9.077 (+/- 0.178)	0.206 (+/- 0.003)	0.872 (+/- 0.004)	0.921 (+/- 0.001)

Still the results are not better than the best performing model.

- It didn't happen here but how could the average do better than the best model???
 - From the perspective of the best estimator (in this case CatBoost), why are you adding on worse estimators??

Here's how this can work:

Example	log reg	rand forest	cat boost	Averaged model
1	✓	✓	✗	✓✓✗ => ✓
2	✓	✗	✓	✓✗✓ => ✓

Example loa rea rand forest cat boost Averaged model

In short, as long as the different models make different mistakes, voting can improve the final predictions.

Why not always do this?

1. fit / predict time.
2. Reduction in interpretability.
3. Reduction in code maintainability (e.g. Netflix prize).

What kind of estimators can we combine?

- You can combine:
 - completely different estimators, or similar estimators.
 - estimators trained on different samples.
 - estimators with different hyperparameter values.

Stacking

- Another type of ensemble is stacking.
- Instead of averaging the outputs of each estimator, instead use their outputs as *inputs to another model*.
- By default for classification, it uses logistic regression.
 - We don't need a complex model here necessarily, more of a weighted average.
 - The features going into the logistic regression are the classifier outputs, *not* the original features!
 - So the number of coefficients = the number of base estimators!

In [78]: 1 `from sklearn.ensemble import StackingClassifier`

The code starts to get too slow here; so we'll remove CatBoost.

In [79]: 1 `classifiers_nocat = classifiers.copy()`
2 `del classifiers_nocat["CatBoost"]`

```
In [80]: 1 stacking_model = StackingClassifier(list(classifiers_nocat.items()))
```

```
In [81]: 1 stacking_model.fit(X_train, y_train);
```

What's going on in here?

- It is doing cross-validation by itself by default (see [documentation \(https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html\)](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html))
 - It is fitting the base estimators on the training fold
 - And the predicting on the validation fold
 - And then fitting the meta-estimator on that output (on the validation fold)

Note that `estimators_` are fitted on the full `X` while `final_estimator_` is trained using cross-validated predictions of the base estimators using `cross_val_predict`.

Here is the input features (`X`) to the meta-model:

```
In [82]: 1 valid_sample = train_df.sample(4, random_state=2).drop(columns=["income"])
```

```
In [83]: 1 r3 = {
  2     name: pipe.predict_proba(valid_sample)
  3     for (name, pipe) in stacking_model.named_estimators_.items()
  4 }
  5 r3
```

```
Out[83]: {'logistic regression': array([[4.33522670e-01, 5.66477330e-01],
   [9.99016977e-01, 9.83023032e-04],
   [8.60803004e-01, 1.39196996e-01],
   [9.95299562e-01, 4.70043768e-03]]),
 'decision tree': array([[1., 0.],
   [1., 0.],
   [1., 0.],
   [1., 0.]]),
 'random forest': array([[0.88, 0.12],
   [1., 0.],
   [0.95, 0.05],
   [1., 0.]]),
 'XGBoost': array([[0.75072813, 0.24927188],
   [0.99450225, 0.00549777],
   [0.95364463, 0.04635534],
   [0.9975675 , 0.00243254]], dtype=float32),
 'LightGBM': array([[0.56629894, 0.43370106],
   [0.99263965, 0.00736035],
   [0.91995188, 0.08004812],
   [0.99629847, 0.00370153]])}
```

- Our meta-model is logistic regression (which it is by default).

- Let's look at the learned coefficients.

```
In [84]: 1 pd.DataFrame(
2     data=stacking_model.final_estimator_.coef_[0],
3     index=classifiers_nocat.keys(),
4     columns=["Coefficient"],
5 )
```

Out[84]:

	Coefficient
logistic regression	0.763242
decision tree	-0.011344
random forest	0.218986
XGBoost	2.022841
LightGBM	3.684328

```
In [85]: 1 stacking_model.final_estimator_.intercept_
```

Out[85]: array([-3.31969134])

- It seems that the LightGBM is being trusted the most.

```
In [86]: 1 stacking_model.predict(test_g50k)
```

Out[86]: array(['>50K', '>50K', '>50K', '<=50K'], dtype=object)

```
In [87]: 1 stacking_model.predict_proba(test_g50k)
```

Out[87]: array([[0.03395918, 0.96604082],
 [0.21342258, 0.78657742],
 [0.22864056, 0.77135944],
 [0.88196532, 0.11803468]])

(This is the `predict_proba` from logistic regression)

Let's see how well this model performs.

```
In [88]: 1 results["Stacking_nocat"] = mean_std_cross_val_scores(
2     stacking_model, X_train, y_train, return_train_score=True, scoring=
3 )
```

In [89]:

```
1 pd.DataFrame(results).T
```

Out[89]:

		fit_time	score_time	test_score	train_score
	Dummy	0.007 (+/- 0.000)	0.006 (+/- 0.000)	0.631 (+/- 0.006)	0.634 (+/- 0.003)
	logistic regression	1.316 (+/- 0.044)	0.010 (+/- 0.000)	0.850 (+/- 0.006)	0.851 (+/- 0.001)
	decision tree	0.144 (+/- 0.002)	0.010 (+/- 0.001)	0.813 (+/- 0.003)	1.000 (+/- 0.000)
	random forest	1.398 (+/- 0.028)	0.082 (+/- 0.002)	0.857 (+/- 0.004)	1.000 (+/- 0.000)
	XGBoost	1.066 (+/- 0.031)	0.015 (+/- 0.000)	0.870 (+/- 0.003)	0.909 (+/- 0.002)
	LightGBM	0.740 (+/- 0.062)	0.016 (+/- 0.001)	0.871 (+/- 0.004)	0.892 (+/- 0.000)
	CatBoost	4.566 (+/- 0.075)	0.079 (+/- 0.001)	0.872 (+/- 0.003)	0.900 (+/- 0.001)
	Voting	9.208 (+/- 0.147)	0.222 (+/- 0.003)	0.868 (+/- 0.003)	NaN
	Voting_ndt	9.077 (+/- 0.178)	0.206 (+/- 0.003)	0.872 (+/- 0.004)	0.921 (+/- 0.001)
	Stacking_nocat	24.434 (+/- 0.534)	0.137 (+/- 0.002)	0.872 (+/- 0.004)	0.900 (+/- 0.007)

- The situation here is a bit mind-boggling.
- On each fold of cross-validation it is doing cross-validation.
- This is really loops within loops within loops...

- We can also try a different final estimator:
- Let's `DecisionTreeClassifier` as a final estimator.

In [90]:

```
1 stacking_model_tree = StackingClassifier(
2     list(classifiers_nocat.items()), final_estimator=DecisionTreeClassifi
3 )
```

The results are not very good. But we can look at the tree:

In [91]:

```
1 stacking_model_tree.fit(X_train, y_train);
```

In [92]:

```
1 display_tree(list(classifiers_nocat.keys()), stacking_model_tree.final_
```

Out[92]:

```
<graphviz.sources.Source at 0x199f7bc10>
```

An effective strategy

- Randomly generate a bunch of models with different hyperparameter configurations, and then stack all the models.
- What is an advantage of ensembling multiple models as opposed to just choosing one of them?
 - You may get a better score.

- What is an disadvantage of ensembling multiple models as opposed to just choosing one of them?
 - Slower, more code maintenance issues

Summary

- You have a number of models in your toolbox now.
- Ensembles are usually pretty effective.
 - Tree-based classifiers are particularly popular and effective on a wide range of problems.
 - But they trade off code complexity and speed for prediction accuracy.
 - Don't forget that hyperparameter optimization multiplies the slowness of the code!
- Stacking is a bit slower than voting, but generally higher accuracy.
 - As a bonus, you get to see the coefficients for each base classifier.
- All the above models have equivalent regression models.

Relevant papers

- [Fernandez-Delgado et al. 2014 \(<http://jmlr.org/papers/volume15/delgado14a/delgado14a.pdf>\)](http://jmlr.org/papers/volume15/delgado14a/delgado14a.pdf)
compared 179 classifiers on 121 datasets:
 - First best class of methods was Random Forest and second best class of methods was (RBF) SVMs.
- If you like to read original papers [here](https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf) (<https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf>) is the original paper on Random Forests by Leo Breiman.

True or False questions on Random Forests (Class discussion)

1. Every tree in a random forest uses a different bootstrap sample of the training set.
2. To train a tree in a random forest, we first randomly select a subset of features. The tree is then restricted to only using those features.
3. A reasonable implementation of `predict_proba` for random forests would be for each tree to "vote" and then normalize these vote counts into probabilities.
4. Increasing the hyperparameter `max_features` (the number of features to consider for a split) makes the model more complex and moves the fundamental tradeoff toward lower training error.
5. A random forest with only one tree is likely to get a higher training error than a decision tree of the same depth.

How would you carry out "soft voting" with `predict_proba` output instead of hard voting for random forests?

Lecture 12: Feature importances

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

```
In [1]: 1 import os
2 import string
3 import sys
4 from collections import deque
5
6 import matplotlib.pyplot as plt
7 import numpy as np
8 import pandas as pd
9
10 sys.path.append("../code/.")
11
12 import seaborn as sns
13 from plotting_functions import *
14 from sklearn import datasets
15 from sklearn.compose import ColumnTransformer, make_column_transformer
16 from sklearn.dummy import DummyClassifier, DummyRegressor
17 from sklearn.ensemble import RandomForestClassifier, RandomForestRegres
18 from sklearn.impute import SimpleImputer
19 from sklearn.linear_model import LogisticRegression, Ridge
20 from sklearn.model_selection import (
21     GridSearchCV,
22     RandomizedSearchCV,
23     cross_val_score,
24     cross_validate,
25     train_test_split,
26 )
27 from sklearn.pipeline import Pipeline, make_pipeline
28 from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, Standar
29 from sklearn.svm import SVC, SVR
30 from sklearn.tree import DecisionTreeClassifier
31 from utils import *
32
33 %matplotlib inline
```

Learning outcomes

From this lecture, students are expected to be able to:

- Interpret the coefficients of linear regression for ordinal, one-hot encoded categorical, and scaled numeric features.

- Explain why interpretability is important in ML.
- Use `feature_importances_` attribute of `sklearn` models and interpret its output.
- Use `eli5` to get feature importances of non `sklearn` models and interpret its output.
- Apply SHAP to assess feature importances and interpret model predictions.
- Explain force plot, summary plot, and dependence plot produced with shapely values.

In [2]:

```
1 import warnings
2
3 warnings.simplefilter(action="ignore", category=FutureWarning)
```

Data

In this lecture, we'll be using [Kaggle House Prices dataset \(<https://www.kaggle.com/c/home-data-for-ml-course>\)](https://www.kaggle.com/c/home-data-for-ml-course), the dataset we used in lecture 2. As usual, to run this notebook you'll need to download the data. Unzip the data into a subdirectory called `data`. For this dataset, train and test have already been separated. We'll be working with the train portion in this lecture.

In [3]:

```
1 df = pd.read_csv("../data/housing-kaggle/train.csv")
2 train_df, test_df = train_test_split(df, test_size=0.10, random_state=1)
3 train_df.head()
```

Out[3]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	L
302	303	20	RL	118.0	13704	Pave	NaN	IR1		Lvl
767	768	50	RL	75.0	12508	Pave	NaN	IR1		Lvl
429	430	20	RL	130.0	11457	Pave	NaN	IR1		Lvl
1139	1140	30	RL	98.0	8731	Pave	NaN	IR1		Lvl
558	559	60	RL	57.0	21872	Pave	NaN	IR2		HLS

5 rows × 81 columns

- The prediction task is predicting `SalePrice` given features related to properties.
- Note that the target is numeric, not categorical (it's a regression problem).

In [4]:

```
1 train_df.shape
```

Out[4]:

```
(1314, 81)
```

Let's separate x and y

```
In [5]: 1 X_train = train_df.drop(columns=["SalePrice"])
2 y_train = train_df["SalePrice"]
3
4 X_test = test_df.drop(columns=["SalePrice"])
5 y_test = test_df["SalePrice"]
```

Let's identify feature types

```
In [6]: 1 drop_features = ["Id"]
2 numeric_features = [
3     "BedroomAbvGr",
4     "KitchenAbvGr",
5     "LotFrontage",
6     "LotArea",
7     "OverallQual",
8     "OverallCond",
9     "YearBuilt",
10    "YearRemodAdd",
11    "MasVnrArea",
12    "BsmtFinSF1",
13    "BsmtFinSF2",
14    "BsmtUnfSF",
15    "TotalBsmtSF",
16    "1stFlrSF",
17    "2ndFlrSF",
18    "LowQualFinSF",
19    "GrLivArea",
20    "BsmtFullBath",
21    "BsmtHalfBath",
22    "FullBath",
23    "HalfBath",
24    "TotRmsAbvGrd",
25    "Fireplaces",
26    "GarageYrBlt",
27    "GarageCars",
28    "GarageArea",
29    "WoodDeckSF",
30    "OpenPorchSF",
31    "EnclosedPorch",
32    "3SsnPorch",
33    "ScreenPorch",
34    "PoolArea",
35    "MiscVal",
36    "YrSold",
37 ]
```

```
In [7]: 1 ordinal_features_reg = [
2     "ExterQual",
3     "ExterCond",
4     "BsmtQual",
5     "BsmtCond",
6     "HeatingQC",
7     "KitchenQual",
8     "FireplaceQu",
9     "GarageQual",
10    "GarageCond",
11    "PoolQC",
12 ]
13 ordering = [
14     "Po",
15     "Fa",
16     "TA",
17     "Gd",
18     "Ex",
19 ] # if N/A it will just impute something, per below
20 ordering_ordinal_reg = [ordering] * len(ordinal_features_reg)
21 ordering_ordinal_reg
```

```
Out[7]: [[ 'Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex']]
```

```
In [8]: 1 ordinal_features_oth = [
2     "BsmtExposure",
3     "BsmtFinType1",
4     "BsmtFinType2",
5     "Functional",
6     "Fence",
7 ]
8 ordering_ordinal_oth = [
9     ["NA", "No", "Mn", "Av", "Gd"],
10    ["NA", "Unf", "LwQ", "Rec", "BLQ", "ALQ", "GLQ"],
11    ["NA", "Unf", "LwQ", "Rec", "BLQ", "ALQ", "GLQ"],
12    ["Sal", "Sev", "Maj2", "Maj1", "Mod", "Min2", "Min1", "Typ"],
13    ["NA", "MnWw", "GdWo", "MnPrv", "GdPrv"],
14 ]
```

```
In [9]: 1 categorical_features = list(  
2     set(X_train.columns)  
3     - set(numeric_features)  
4     - set(ordinal_features_reg)  
5     - set(ordinal_features_oth)  
6     - set(drop_features)  
7 )  
8 categorical_features
```

```
Out[9]: ['Electrical',  
'LotShape',  
'Exterior1st',  
'MiscFeature',  
'LandContour',  
'RoofMatl',  
'Foundation',  
'MSZoning',  
'LandSlope',  
'SaleType',  
'Street',  
'HouseStyle',  
'Condition1',  
'GarageFinish',  
'Heating',  
'Neighborhood',  
'Exterior2nd',  
'Condition2',  
'PavedDrive',  
'MSSubClass',  
'Utilities',  
'Alley',  
'MasVnrType',  
'RoofStyle',  
'SaleCondition',  
'BldgType',  
'CentralAir',  
'MoSold',  
'LotConfig',  
'GarageType']
```

In [10]:

```
1 from sklearn.compose import ColumnTransformer, make_column_transformer
2
3 numeric_transformer = make_pipeline(SimpleImputer(strategy="median"), S
4 ordinal_transformer_reg = make_pipeline(
5     SimpleImputer(strategy="most_frequent"),
6     OrdinalEncoder(categories=ordering_ordinal_reg),
7 )
8
9 ordinal_transformer_oth = make_pipeline(
10    SimpleImputer(strategy="most_frequent"),
11    OrdinalEncoder(categories=ordering_ordinal_oth),
12 )
13
14 categorical_transformer = make_pipeline(
15    SimpleImputer(strategy="constant", fill_value="missing"),
16    OneHotEncoder(handle_unknown="ignore", sparse=False),
17 )
18
19 preprocessor = make_column_transformer(
20     ("drop", drop_features),
21     (numeric_transformer, numeric_features),
22     (ordinal_transformer_reg, ordinal_features_reg),
23     (ordinal_transformer_oth, ordinal_features_oth),
24     (categorical_transformer, categorical_features),
25 )
```

```
In [11]: 1 preprocessor.fit(X_train)
          2 preprocessor.named_transformers_
```

```
Out[11]: {'drop': 'drop',
  'pipeline-1': Pipeline(steps=[('simpleimputer', SimpleImputer(strategy='median')),
                                 ('standardscaler', StandardScaler()))]),
  'pipeline-2': Pipeline(steps=[('simpleimputer', SimpleImputer(strategy='most_frequent')),
                                 ('ordinalencoder',
                                  OrdinalEncoder(categories=[[ 'Po', 'Fa', 'TA', 'Gd', 'E
x'],
                                                               [ 'Po', 'Fa', 'TA', 'Gd', 'E
x']])),
  'pipeline-3': Pipeline(steps=[('simpleimputer', SimpleImputer(strategy='most_frequent')),
                                 ('ordinalencoder',
                                  OrdinalEncoder(categories=[[ 'NA', 'No', 'Mn', 'Av', 'G
d'],
                                                               [ 'NA', 'Unf', 'LwQ', 'Rec',
                                                               'BLQ'],
                                                               [ 'NA', 'Unf', 'LwQ', 'Rec',
                                                               'BLQ'],
                                                               [ 'ALQ', 'GLQ'],
                                                               [ 'Sal', 'Sev', 'Maj2', 'Maj
1'],
                                                               [ 'Mod', 'Min2', 'Min1', 'Ty
p'],
                                                               [ 'NA', 'MnWw', 'GdWo', 'MnPr
v'],
                                                               [ 'GdPrv']])))],
  'pipeline-4': Pipeline(steps=[('simpleimputer',
                                 SimpleImputer(fill_value='missing', strategy='constan
t')),
                                 ('onehotencoder',
                                  OneHotEncoder(handle_unknown='ignore', sparse=False))])}
```

```
In [12]: 1 ohe_columns = list(
2     preprocessor.named_transformers_["pipeline-4"]
3     .named_steps["onehotencoder"]
4     .get_feature_names(categorical_features)
5 )
6 new_columns = (
7     numeric_features + ordinal_features_reg + ordinal_features_oth + oh
8 )
```

```
In [13]: 1 X_train_enc = pd.DataFrame(
2     preprocessor.transform(X_train), index=X_train.index, columns=new_c
3 )
4 X_train_enc
```

Out[13]:

	BedroomAbvGr	KitchenAbvGr	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	Y
302	0.154795	-0.222647	2.312501	0.381428	0.663680	-0.512408	0.993969	
767	1.372763	-0.222647	0.260890	0.248457	-0.054669	1.285467	-1.026793	
429	0.154795	-0.222647	2.885044	0.131607	-0.054669	-0.512408	0.563314	
1139	0.154795	-0.222647	1.358264	-0.171468	-0.773017	-0.512408	-1.689338	
558	0.154795	-0.222647	-0.597924	1.289541	0.663680	-0.512408	0.828332	
...
1041	1.372763	-0.222647	-0.025381	-0.127107	-0.054669	2.184405	-0.165485	
1122	0.154795	-0.222647	-0.025381	-0.149788	-1.491366	-2.310284	-0.496757	
1346	0.154795	-0.222647	-0.025381	1.168244	0.663680	1.285467	-0.099230	
1406	-1.063173	-0.222647	0.022331	-0.203265	-0.773017	1.285467	0.033279	
1389	0.154795	-0.222647	-0.454788	-0.475099	-0.054669	0.386530	-0.993666	

1314 rows × 263 columns

```
In [14]: 1 X_train_enc.shape
```

```
Out[14]: (1314, 263)
```

Feature correlations

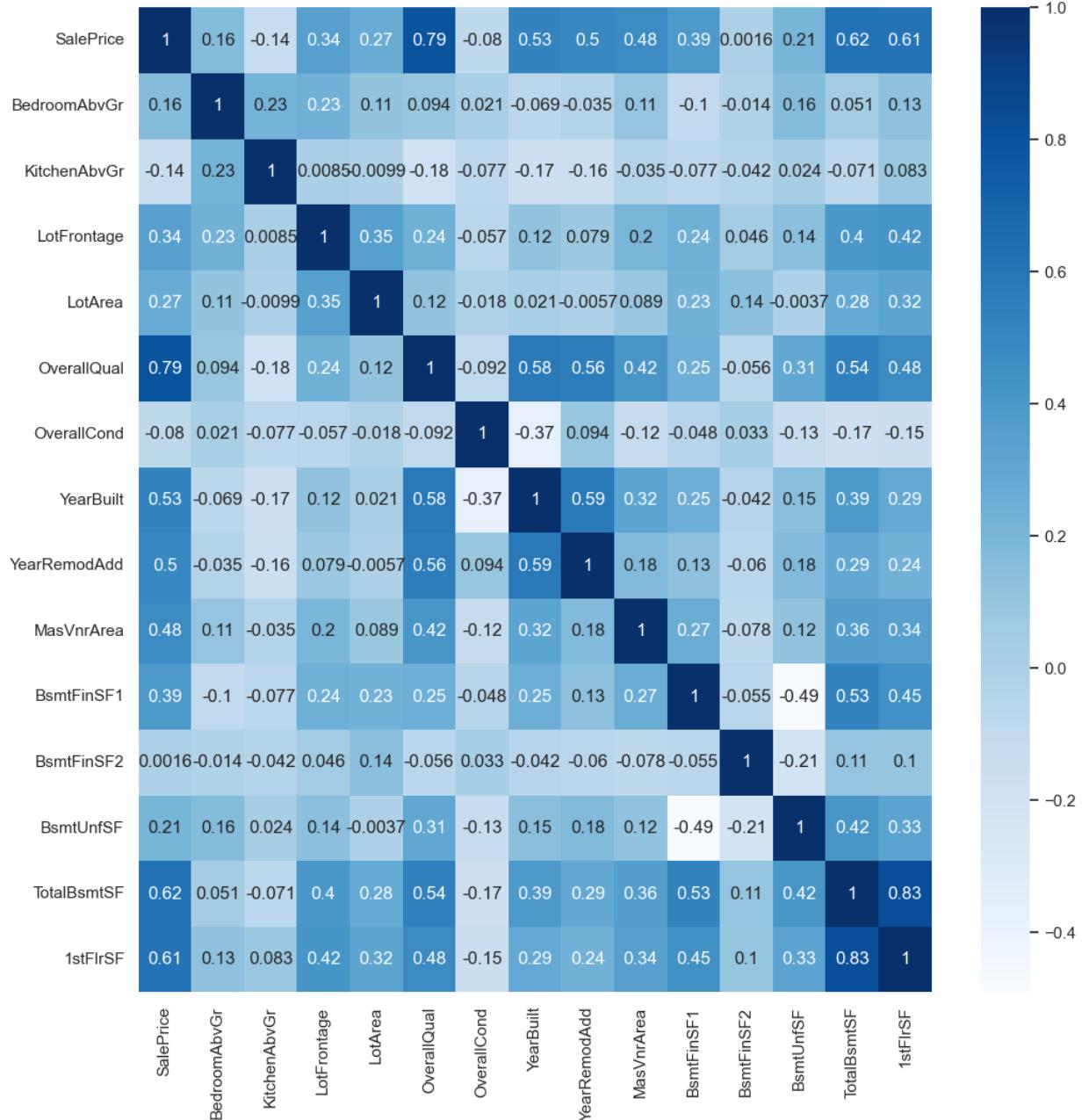
- Let's look at the correlations between various features with other features and the target in our encoded data.
- In simple terms here is how you can interpret correlations between two variables X and Y :
 - If Y goes up when X goes up, we say X and Y are positively correlated.
 - If Y goes down when X goes up, we say X and Y are negatively correlated.
 - If Y does not change in predictable ways when X changes, we say X and Y are uncorrelated.

In [15]:

```

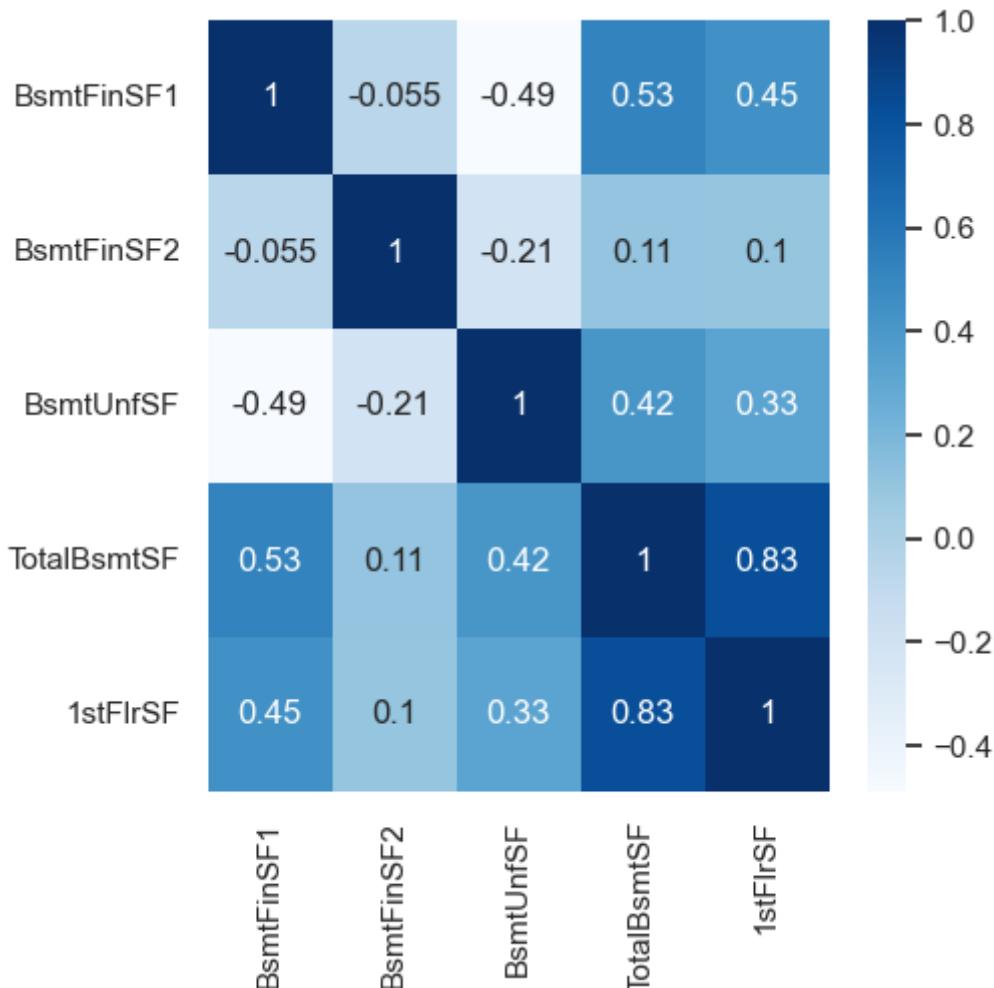
1 cor = pd.concat((y_train, X_train_enc), axis=1).iloc[:, :15].corr()
2 plt.figure(figsize=(12, 12))
3 sns.set(font_scale=1)
4 sns.heatmap(cor, annot=True, cmap=plt.cm.Blues);

```



- We can immediately see that `SalePrice` is highly correlated with `OverallQual`.
- This is an early hint that `OverallQual` is a useful feature in predicting `SalePrice`.
- However, this approach is **extremely simplistic**.
 - It only looks at each feature in isolation.
 - It only looks at linear associations:
 - What if `SalePrice` is high when `BsmtFullBath` is 2 or 3, but low when it's 0, 1, or 4? They might seem uncorrelated.

```
In [16]: 1 cor = pd.concat((y_train, X_train_enc), axis=1).iloc[:, 10:15].corr()
2 plt.figure(figsize=(5, 5))
3 sns.set(font_scale=1)
4 sns.heatmap(cor, annot=True, cmap=plt.cm.Blues);
```



- Looking at this diagram also tells us the relationship between features.
 - For example, `1stFlrSF` and `TotalBsmtSF` are highly correlated.
 - Do we need both of them?
 - If our model says `1stFlrSF` is very important and `TotalBsmtSF` is very unimportant, do we trust those values?
 - Maybe `TotalBsmtSF` only "becomes important" if `1stFlrSF` is removed.
 - Sometimes the opposite happens: a feature only becomes important if another feature is added.

Feature importances in linear models

- Like logistic regression, with linear regression we can look at the *coefficients* for each feature.
- Overall idea: predicted price = intercept + \sum_i coefficient i \times feature i.

```
In [17]: 1 lr = make_pipeline(preprocessor, Ridge())
2 lr.fit(X_train, y_train);
```

Let's look at the coefficients.

```
In [18]: 1 lr_coefs = pd.DataFrame(data=lr[1].coef_, index=new_columns, columns=["])
2 lr_coefs.head(20)
```

	Coefficient
BedroomAbvGr	-3723.741570
KitchenAbvGr	-4580.204576
LotFrontage	-1578.664421
LotArea	5109.356718
OverallQual	12487.561839
OverallCond	4855.535334
YearBuilt	4226.684842
YearRemodAdd	324.664715
MasVnrArea	5251.325210
BsmtFinSF1	3667.172851
BsmtFinSF2	583.114880
BsmtUnfSF	-1266.614671
TotalBsmtSF	2751.084018
1stFlrSF	6736.788904
2ndFlrSF	13409.901084
LowQualFinSF	-448.424132
GrLivArea	15988.182407
BsmtFullBath	2299.227266
BsmtHalfBath	500.169112
FullBath	2831.811467

Interpreting coefficients of different types of features.

Ordinal features

- The ordinal features are easiest to interpret.

```
In [19]: 1 print(ordinal_features_reg)
```

```
['ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCond', 'HeatingQC', 'KitchenQual', 'FireplaceQu', 'GarageQual', 'GarageCond', 'PoolQC']
```

```
In [20]: 1 lr_coefs.loc["ExterQual", "Coefficient"]
```

```
Out[20]: 4195.671512467474
```

- Increasing by one category of exterior quality (e.g. good -> excellent) increases the predicted price by ~ \$4195.
 - Wow, that's a lot!
 - Remember this is just what the model has learned. It doesn't tell us how the world works.

```
In [21]: 1 one_example = X_test[:1]
```

```
In [22]: 1 one_example["ExterQual"]
```

```
Out[22]: 147      Gd
Name: ExterQual, dtype: object
```

Let's perturb the example and change `ExterQual` to `Ex`.

```
In [23]: 1 one_example_perturbed = one_example.copy()
2 one_example_perturbed["ExterQual"] = "Ex" # Change Gd to Ex
```

```
In [24]: 1 one_example_perturbed
```

```
Out[24]:
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Util	
	147	148	60	RL	NaN	9505	Pave	NaN	IR1	Lvl	Al

1 rows × 80 columns

```
In [25]: 1 one_example_perturbed["ExterQual"]
```

```
Out[25]: 147      Ex
Name: ExterQual, dtype: object
```

How does the prediction change after changing `ExterQual` from `Gd` to `Ex`?

```
In [26]: 1 print("Prediction on the original example: ", lr.predict(one_example))
2 print("Prediction on the perturbed example: ", lr.predict(one_example_p
3 print(
4     "After changing ExterQual from Gd to Ex increased the prediction by
5     lr.predict(one_example_perturbed) - lr.predict(one_example),
6 )
```

```
Prediction on the original example: [224795.63596803]
Prediction on the perturbed example: [228991.30748049]
After changing ExterQual from Gd to Ex increased the prediction by: [419
5.67151247]
```

That's exactly the learned coefficient for ExterQual !

```
In [27]: 1 lr_coefs.loc["ExterQual", "Coefficient"]
```

```
Out[27]: 4195.671512467474
```

So our interpretation is correct!

- Increasing by one category of exterior quality (e.g. good -> excellent) increases the predicted price by ~ \$4195.

Categorical features

- What about the categorical features?
- We have created a number of columns for each category with OHE and each category gets its own coefficient.

```
In [28]: 1 print(categorical_features)
```

```
['Electrical', 'LotShape', 'Exterior1st', 'MiscFeature', 'LandContour',
'RoofMatl', 'Foundation', 'MSZoning', 'Landslope', 'SaleType', 'Street',
'HouseStyle', 'Condition1', 'GarageFinish', 'Heating', 'Neighborhood', 'E
xterior2nd', 'Condition2', 'PavedDrive', 'MSSubClass', 'Utilities', 'Alle
y', 'MasVnrType', 'RoofStyle', 'SaleCondition', 'BldgType', 'CentralAir',
'MoSold', 'LotConfig', 'GarageType']
```

```
In [29]: 1 lr_coefs_landslope = lr_coefs[lr_coefs.index.str.startswith("LandSlope")]
2 lr_coefs_landslope
```

	Coefficient
LandSlope_Gtl	457.197456
LandSlope_Mod	7420.208381
LandSlope_Sev	-7877.405837

- We can talk about switching from one of these categories to another by picking a "reference" category:

```
In [30]: 1 lr_coefs_landslope = lr_coefs_landslope.loc["LandSlope_Gtl"]
```

Out[30]:

	Coefficient
LandSlope_Gtl	0.000000
LandSlope_Mod	6963.010925
LandSlope_Sev	-8334.603292

- If you change the category from `LandSlope_Gtl` to `LandSlope_Mod` the prediction price goes up by ~ \$6963
- If you change the category from `LandSlope_Gtl` to `LandSlope_Sev` the prediction price goes down by ~ \$8334

Note that this might not make sense in the real world but this is what our model decided to learn given this small amount of data.

```
In [31]: 1 lr_coefs.sort_values(by="Coefficient")
```

Out[31]:

	Coefficient
RoofMatl_ClyTile	-191129.774314
Condition2_PosN	-105552.840565
Heating_OthW	-27260.681308
MSZoning_C (all)	-21990.746193
Exterior1st_ImStucc	-19393.964621
...	...
PoolQC	34217.656047
RoofMatl_CompShg	36525.980874
Neighborhood_NridgHt	37532.643270
Neighborhood_StoneBr	39993.978324
RoofMatl_WdShngl	83646.711008

263 rows × 1 columns

- For example, the above coefficient says that "If the roof is made of clay or tile, the predicted price is \$191K less"?
- Do we believe these interpretations???
 - Do we believe this is how the predictions are being computed? Yes.
 - Do we believe that this is how the world works? No.

If you did `drop='first'` (we didn't) then you already have a reference class, and all the values are with respect to that one. **The interpretation depends on the variable encoding**, here whether we did `drop='first'`.

Interpreting coefficients of numeric features

Let's look at coefficients of `PoolArea` and `LotFrontage`.

```
In [32]: 1 lr_coefs.loc[[ "PoolArea", "LotArea"]]
```

```
Out[32]:
```

	Coefficient
PoolArea	2822.370476
LotArea	5109.356718

Intuition:

- Tricky because numeric features are **scaled!**
- **Increasing** `PoolArea` by 1 scaled unit **increases** the predicted price by $\sim \$2822$.
- **Increasing** `LotFrontage` by 1 scaled unit **decreases** the predicted price by $\sim \$1578$.

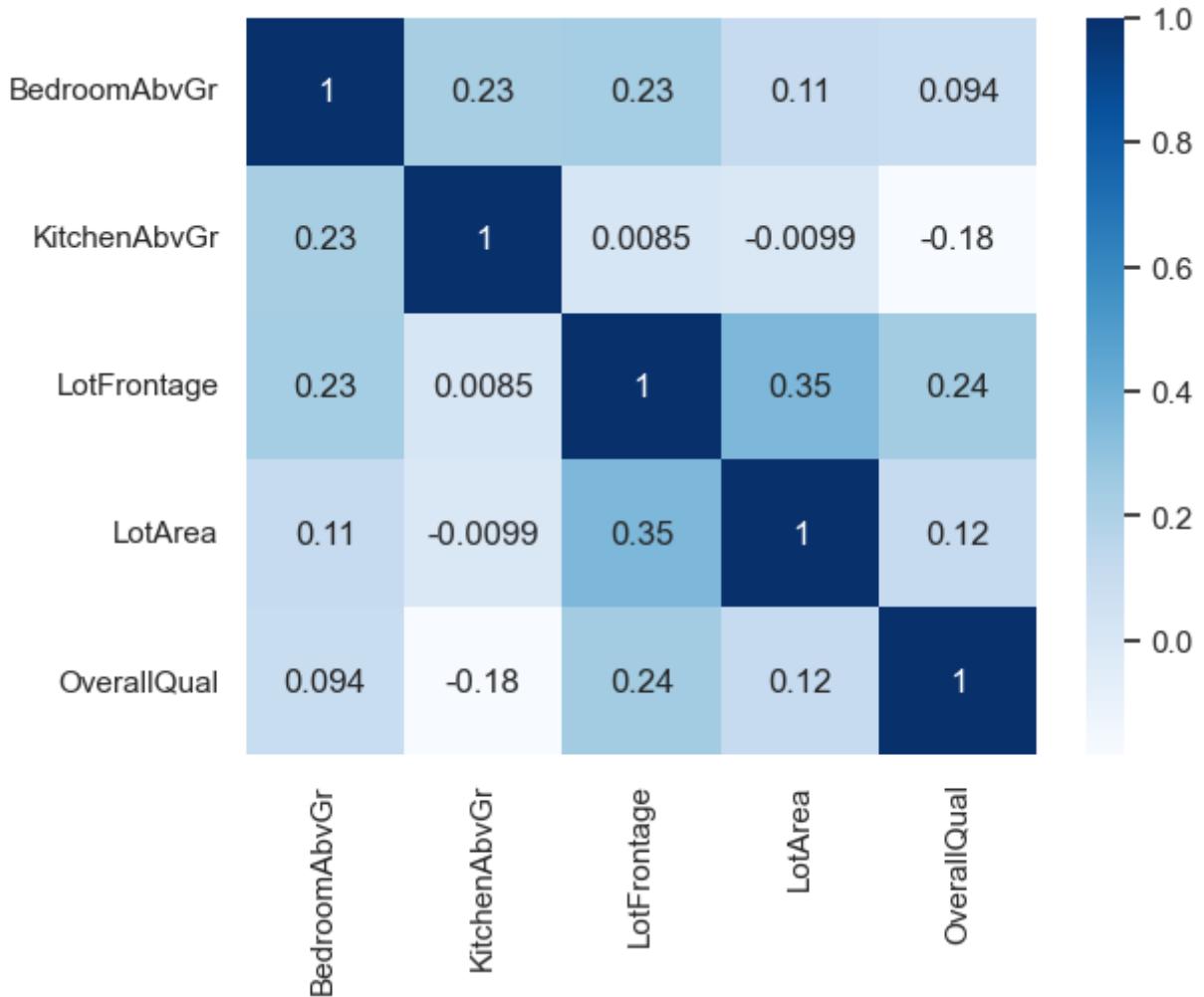
Does that sound reasonable?

- For `PoolArea`, yes.
- For `LotFrontage`, that's surprising. Something positive would have made more sense?

It's not the case here but maybe the problem is that `LotFrontage` and `LotArea` are very correlated. `LotArea` has a larger positive coefficient.

In [33]:

```
1 cor = X_train_enc[numeric_features[:5]].corr()
2 sns.heatmap(cor, annot=True, cmap=plt.cm.Blues);
```



BTW, let's make sure the predictions behave as expected:

In [34]:

```
1 one_example = X_test[:1]
```

In [35]:

```
1 one_example
```

Out[35]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Util	
	147	148	60	RL	NaN	9505	Pave	NaN	IR1	Lvl	Al

1 rows × 80 columns

Let's perturb the example and add 1 to the `LotArea`.

In [36]:

```
1 one_example_perturbed = one_example.copy()
2 one_example_perturbed["LotArea"] += 1 # add 1 to the LotArea
```

In [37]: 1 one_example_perturbed

Out[37]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Util
147	148	60	RL	NaN	9506	Pave	NaN	IR1		Lvl Al

1 rows × 80 columns

Prediction on the original example.

In [38]: 1 lr.predict(one_example)

Out[38]: array([224795.63596803])

Prediction on the perturbed example.

In [39]: 1 lr.predict(one_example_perturbed)

Out[39]: array([224796.2040233])

- What's the difference between prediction?
- Does the difference make sense given the coefficient of the feature?

In [40]: 1 lr.predict(one_example_perturbed) - lr.predict(one_example)

Out[40]: array([0.56805528])

In [41]: 1 lr_coefs.loc[["LotArea"]]

Out[41]:

	Coefficient
LotArea	5109.356718

- Why did the prediction only go up by \$0.57 instead of \$5109?
- The scaling happens **after our change**, so this is an issue of units: `LotArea` is in sqft, but the coefficient is not \$5109/sqft **because we scaled the features**.

Example showing how to interpret coefficients of scaled features.

- The scaler subtracted the mean and divided by the standard deviation.
- The division actually changed the scale!
- For the unit conversion, we don't care about the subtraction, but only the scaling.

In [42]: 1 scaler = preprocessor.named_transformers_["pipeline-1"]["standardscaler"]

```
In [43]: 1 np.sqrt(scaler.var_)
```

```
Out[43]: array([8.21039683e-01, 2.18760172e-01, 2.09591390e+01, 8.99447103e+03,
   1.39208177e+00, 1.11242416e+00, 3.01866337e+01, 2.06318985e+01,
   1.77914527e+02, 4.59101890e+02, 1.63890010e+02, 4.42869860e+02,
   4.42817167e+02, 3.92172897e+02, 4.35820743e+02, 4.69800920e+01,
   5.29468070e+02, 5.18276015e-01, 2.33809970e-01, 5.49298599e-01,
   5.02279069e-01, 1.62604030e+00, 6.34398801e-01, 2.40531598e+01,
   7.40269201e-01, 2.10560601e+02, 1.25388753e+02, 6.57325181e+01,
   6.07432962e+01, 3.03088902e+01, 5.38336322e+01, 4.23249944e+01,
   5.22084645e+02, 1.33231649e+00])
```

```
In [44]: 1 lr_scales = pd.DataFrame(
   2     data=np.sqrt(scaler.var_), index=numeric_features, columns=["Scale"]
   3 )
   4 lr_scales.head()
```

	Scale
BedroomAbvGr	0.821040
KitchenAbvGr	0.218760
LotFrontage	20.959139
LotArea	8994.471032
OverallQual	1.392082

- It seems like `LotArea` was divided by 8994.471032 sqft.

```
In [45]: 1 lr_coefs.loc["LotArea", "Coefficient"]
```

```
Out[45]: 5109.356718094066
```

```
In [46]: 1 lr_coefs.loc["LotArea", "Coefficient"] / lr_scales.loc["LotArea", "Scale"]
```

```
Out[46]: 0.5680552752646618
```

```
In [47]: 1 lr_coefs.loc[[ "LotArea"]]
```

```
Out[47]:
```

Coefficient
LotArea 5109.356718

- The coefficient tells us that if we increase the **scaled** `LotArea` by one unit the price would go up by $\approx \$5109$.
- One scaled unit represents ~ 8994 sq feet.

- So if I increase original `LotArea` by one square foot then the predicted price would go up by this amount:

In [48]: 1 5109.356718094072 / 8994.471032

Out[48]: 0.5680552752814816

- This makes much more sense. Now we get the number we got before.
- That said don't read too much into these coefficients without statistical training.

Interim summary

- Correlation among features might make coefficients completely uninterpretable.
- Fairly straightforward to interpret coefficients of ordinal features.
- In categorical features, it's often helpful to consider one category as a reference point and think about relative importance.
- For numeric features, relative importance is meaningful after scaling.
- You have to be careful about the scale of the feature when interpreting the coefficients.
- Remember that explaining the model \neq explaining the data.
- the coefficients tell us only about the model and they might not accurately reflect the data.

Break (5 min)

Interpretability of ML models: Motivations

Why model interpretability?

- Ability to interpret ML models is crucial in many applications such as banking, healthcare, and criminal justice.
- It can be leveraged by domain experts to diagnose systematic errors and underlying biases of complex ML systems.

Source

(<https://github.com/slundberg/shap/blob/master/docs/presentations/February%202018%20Talk.ppt>)

What is model interpretability?

- In this course, our definition of model interpretability will be looking at **feature importances**.
- There is more to interpretability than feature importances, but it's a good start!

- Resource:

- [Interpretable Machine Learning \(<https://christophm.github.io/interpretable-ml-book/interpretability-importance.html>\)](https://christophm.github.io/interpretable-ml-book/interpretability-importance.html)
- [Yann LeCun, Kilian Weinberger, Patrice Simard, and Rich Caruana: Panel debate on interpretability \(<https://vimeo.com/252187813>\)](https://vimeo.com/252187813)

Data

- Let's work with [the adult census data set \(<https://www.kaggle.com/uciml/adult-census-income>\)](https://www.kaggle.com/uciml/adult-census-income) from last lecture.

In [49]:

```

1 adult_df_large = pd.read_csv("../data/adult.csv")
2 train_df, test_df = train_test_split(adult_df_large, test_size=0.2, ran
3 train_df_nan = train_df.replace("?", np.NaN)
4 test_df_nan = test_df.replace("?", np.NaN)
5 train_df_nan.head()

```

Out[49]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship
5514	26	Private	256263	HS-grad		9	Never-married	Craft-repair
19777	24	Private	170277	HS-grad		9	Never-married	Other-service
10781	36	Private	75826	Bachelors		13	Divorced	Adm-clerical
32240	22	State-gov	24395	Some-college		10	Married-civ-spouse	Adm-clerical
9876	31	Local-gov	356689	Bachelors		13	Married-civ-spouse	Prof-specialty

In [50]:

```

1 numeric_features = ["age", "fnlwgt", "capital.gain", "capital.loss", "h
2 categorical_features = [
3     "workclass",
4     "marital.status",
5     "occupation",
6     "relationship",
7     "native.country",
8 ]
9 ordinal_features = ["education"]
10 binary_features = ["sex"]
11 drop_features = ["race", "education.num"]
12 target_column = "income"

```

```
In [51]: 1 education_levels = [
2     "Preschool",
3     "1st-4th",
4     "5th-6th",
5     "7th-8th",
6     "9th",
7     "10th",
8     "11th",
9     "12th",
10    "HS-grad",
11    "Prof-school",
12    "Assoc-voc",
13    "Assoc-acdm",
14    "Some-college",
15    "Bachelors",
16    "Masters",
17    "Doctorate",
18 ]
```

```
In [52]: 1 assert set(education_levels) == set(train_df["education"].unique())
```

```
In [53]: 1 numeric_transformer = make_pipeline(SimpleImputer(strategy="median"), S
2 tree_numeric_transformer = make_pipeline(SimpleImputer(strategy="median"),
3
4 categorical_transformer = make_pipeline(
5     SimpleImputer(strategy="constant", fill_value="missing"),
6     OneHotEncoder(handle_unknown="ignore"),
7 )
8
9 ordinal_transformer = make_pipeline(
10    SimpleImputer(strategy="constant", fill_value="missing"),
11    OrdinalEncoder(categories=[education_levels], dtype=int),
12 )
13
14 binary_transformer = make_pipeline(
15    SimpleImputer(strategy="constant", fill_value="missing"),
16    OneHotEncoder(drop="if_binary", dtype=int),
17 )
18
19 preprocessor = make_column_transformer(
20     ("drop", drop_features),
21     (numeric_transformer, numeric_features),
22     (ordinal_transformer, ordinal_features),
23     (binary_transformer, binary_features),
24     (categorical_transformer, categorical_features),
25 )
```

```
In [54]: 1 X_train = train_df_nan.drop(columns=[target_column])
2 y_train = train_df_nan[target_column]
3
4 X_test = test_df_nan.drop(columns=[target_column])
5 y_test = test_df_nan[target_column]
```

Do we have class imbalance?

- There is class imbalance. But without any context, both classes seem equally important.
- Let's use accuracy as our metric.

```
In [55]: 1 train_df_nan["income"].value_counts(normalize=True)
```

```
Out[55]: <=50K    0.757985
>50K    0.242015
Name: income, dtype: float64
```

```
In [56]: 1 scoring_metric = "accuracy"
```

Let's store all the results in a dictionary called `results`.

```
In [57]: 1 results = {}
```

```
In [58]: 1 scoring_metric = "accuracy"
```

```
In [59]: 1 from lightgbm.sklearn import LGBMClassifier
2 from xgboost import XGBClassifier
3 from sklearn.preprocessing import LabelEncoder
4
5 pipe_lr = make_pipeline(
6     preprocessor, LogisticRegression(max_iter=2000, random_state=123)
7 )
8 pipe_rf = make_pipeline(preprocessor, RandomForestClassifier(random_st
9 pipe_xgb = make_pipeline(
10     preprocessor, XGBClassifier(random_state=123, eval_metric="logloss"
11 )
12 pipe_lgbm = make_pipeline(preprocessor, LGBMClassifier(random_state=123
13
14 # XGBoost requires numeric targets
15 label_encoder = LabelEncoder()
16 label_encoder.fit(y_train)
17
18 classifiers = {
19     "logistic regression": pipe_lr,
20     "random forest": pipe_rf,
21     "XGBoost": pipe_xgb,
22     "LightGBM": pipe_lgbm,
23 }
```

```
In [60]: 1 dummy = DummyClassifier(strategy="stratified")
2 results["Dummy"] = mean_std_cross_val_scores(
3     dummy, X_train, y_train, return_train_score=True, scoring=scoring_m
4 )
```

```
In [61]: 1 for (name, model) in classifiers.items():
2     results[name] = mean_std_cross_val_scores(
3         model, X_train, label_encoder.transform(y_train), return_train_
4     )
```

```
In [62]: 1 pd.DataFrame(results).T
```

Out[62]:

	fit_time	score_time	test_score	train_score
Dummy	0.008 (+/- 0.001)	0.005 (+/- 0.000)	0.631 (+/- 0.002)	0.633 (+/- 0.002)
logistic regression	0.633 (+/- 0.056)	0.010 (+/- 0.000)	0.850 (+/- 0.006)	0.851 (+/- 0.001)
random forest	6.678 (+/- 0.022)	0.074 (+/- 0.001)	0.857 (+/- 0.004)	1.000 (+/- 0.000)
XGBoost	0.917 (+/- 0.126)	0.017 (+/- 0.001)	0.871 (+/- 0.004)	0.908 (+/- 0.001)
LightGBM	0.771 (+/- 0.086)	0.018 (+/- 0.001)	0.871 (+/- 0.004)	0.892 (+/- 0.000)

- One problem is that often simple models are interpretable but not accurate.
- But more complex models (e.g., LightGBM) are less interpretable.

Source

(<https://github.com/slundberg/shap/blob/master/docs/presentations/February%202018%20Talk.ppt>)

Feature importances in linear models

- Simpler models are often more interpretable but less accurate.

Let's create and fit a pipeline with preprocessor and logistic regression.

```
In [63]: 1 pipe_lr = make_pipeline(preprocessor, LogisticRegression(max_iter=2000,
2 pipe_lr.fit(X_train, y_train);
```

```
In [64]: 1 ohe_feature_names = (
2     pipe_rf.named_steps["columntransformer"]
3     .named_transformers_["pipeline-4"]
4     .named_steps["onehotencoder"]
5     .get_feature_names()
6     .tolist()
7 )
8 feature_names = (
9     numeric_features + ordinal_features + binary_features + ohe_feature
10 )
11 feature_names[:10]
```

```
Out[64]: ['age',
'fnlwgt',
'capital.gain',
'capital.loss',
'hours.per.week',
'education',
'sex',
'x0_Federal-gov',
'x0_Local-gov',
'x0_Never-worked']
```

```
In [65]: 1 data = {
2     "coefficient": pipe_lr.named_steps["logisticregression"].coef_[0].t
3     "magnitude": np.absolute(
4         pipe_lr.named_steps["logisticregression"].coef_[0].tolist()
5     ),
6 }
7 coef_df = pd.DataFrame(data, index=feature_names).sort_values(
8     "magnitude", ascending=False
9 )
```

```
In [66]: 1 coef_df[:10]
```

	coefficient	magnitude
capital.gain	2.355403	2.355403
x1_Married-AF-spouse	1.729264	1.729264
x2_Priv-house-serv	-1.408362	1.408362
x1_Married-civ-spouse	1.324596	1.324596
x3_Wife	1.261245	1.261245
x4_Columbia	-1.104853	1.104853
x2_Prof-specialty	1.064275	1.064275
x2_Exec-managerial	1.041384	1.041384
x3_Own-child	-1.014115	1.014115
x4_Dominican-Republic	-1.007272	1.007272

- Increasing `capital.gain` is likely to push the prediction towards ">50k" income class

- Whereas occupation of private house service is likely to push the prediction towards "<=50K" income.

Can we get feature importances for non-linear models?

Model interpretability beyond linear models

We will be looking at three ways for model interpretability.

- `sklearn feature_importances_`
- [eli5 \(<https://eli5.readthedocs.io/en/latest/tutorials/black-box-text-classifiers.html#lime-tutorial>\)](https://eli5.readthedocs.io/en/latest/tutorials/black-box-text-classifiers.html#lime-tutorial)
(stands for "explain like I'm 5")
- [SHAP \(<https://github.com/slundberg/shap>\)](https://github.com/slundberg/shap).

`sklearn feature_importances_`

- Many `sklearn` models have `feature_importances_` attribute.
- For tree-based models it's calculated based on impurity (gini index or information gain).
- For example, let's look at `feature_importances_` of `RandomForestClassifier`.

Let's create and fit a pipeline with preprocessor and random forest.

In [67]:

```
1 pipe_rf = make_pipeline(preprocessor, RandomForestClassifier(random_st
```

Which features are driving the predictions the most?

In [68]:

```

1 data = {
2     "Importance": pipe_rf.named_steps["randomforestclassifier"].feature_
3 }
4 imps = pd.DataFrame(data=data, index=feature_names,).sort_values(
5     by="Importance", ascending=False
6 )[:10]
7 imps

```

Out[68]:

	Importance
fnlwgt	0.169580
age	0.153339
education	0.102953
capital.gain	0.097686
hours.per.week	0.085583
x1_Married-civ-spouse	0.064646
x3_Husband	0.048896
capital.loss	0.033387
x1_Never-married	0.028629
x2_Exec-managerial	0.020458

Key point

- Unlike the linear model coefficients, `feature_importances_` do not have a sign!
 - They tell us about importance, but not an "up or down".
 - Indeed, increasing a feature may cause the prediction to first go up, and then go down.
 - This cannot happen in linear models, because they are linear.

Do these importances match with importances identified by logistic regression?

In [69]:

```

1 data = {
2     "random forest importance": pipe_rf.named_steps[
3         "randomforestclassifier"
4     ].feature_importances_,
5     "logistic regression importances": pipe_lr.named_steps["logisticreg"
6         .coef_[0]
7         .tolist(),
8     ]
9     imps = pd.DataFrame(
10         data=data,
11         index=feature_names,
12     )

```

```
In [70]: 1 imps.sort_values(by="random forest importance", ascending=False)[:10]
```

Out[70]:

	random forest importance	logistic regression importances
fnlwgt	0.169580	0.078087
age	0.153339	0.359883
education	0.102953	0.183963
capital.gain	0.097686	2.355403
hours.per.week	0.085583	0.370353
x1_Married-civ-spouse	0.064646	1.324596
x3_Husband	0.048896	-0.032775
capital.loss	0.033387	0.281123
x1_Never-married	0.028629	-0.956018
x2_Exec-managerial	0.020458	1.041384

- Both models agree on `age`, `education`, `capital.gain`
- The actual numbers for random forests and logistic regression are not really comparable.

How can we get feature importances for non `sklearn` models?

- One way to do it is by using a tool called `eli5` (<https://eli5.readthedocs.io/en/latest/overview.html>).

You'll have to install it

```
conda install -c conda-forge eli5
```

Let's look at feature importances for `XGBClassifier`.

In [71]:

```

1 import eli5
2
3 pipe_xgb = make_pipeline(preprocessor, XGBClassifier(random_state=123,
4 pipe_xgb.fit(X_train, label_encoder.transform(y_train));
5 eli5.explain_weights(pipe_xgb.named_steps["xgbclassifier"], feature_nam

```

Out[71]:

Weight	Feature
0.4061	x1_Married-civ-spouse
0.0547	capital.gain
0.0441	x3_Own-child
0.0349	education
0.0325	x2_Other-service
0.0268	capital.loss
0.0247	x2_Prof-specialty
0.0179	x2_Exec-managerial
0.0178	x2_Tech-support
0.0172	x2_Handlers-cleaners
0.0164	x2_Machine-op-inspct
0.0164	x2_Farming-fishing
0.0158	x0_Federal-gov
0.0117	age
0.0108	x0_Self-emp-inc
0.0107	hours.per.week
0.0102	x3_Wife
0.0101	sex
0.0094	x3_Not-in-family
0.0091	x0_Self-emp-not-inc
... 66 more ...	

Let's look at feature importances for LGBMClassifier .

In [72]:

```

1 pipe_lgbm = make_pipeline(preprocessor, LGBMClassifier(random_state=123
2 pipe_lgbm.fit(X_train, y_train)
3 eli5.explain_weights(
4     pipe_lgbm.named_steps["lgbmclassifier"], feature_names=feature_name
5 )

```

Out[72]:

Weight	Feature
0.3558	x1_Married-civ-spouse
0.1910	capital.gain
0.1363	education
0.0852	age
0.0639	capital.loss
0.0418	hours.per.week
0.0245	fnlwgt
0.0134	x2_Exec-managerial
0.0120	x2_Prof-specialty
0.0067	x2_Other-service
0.0065	sex
0.0055	x3_Wife
0.0054	x0_Self-emp-not-inc
0.0052	x2_Farming-fishing
0.0046	x3_Own-child
0.0033	x2_Tech-support
0.0025	x2_Sales
0.0024	x0_Private
0.0024	x0_Federal-gov
0.0023	x2_Handlers-cleaners
... 66 more ...	

You can also look at feature importances for RandomForestClassifier .

```
In [73]: 1 eli5.explain_weights(
2     pipe_rf.named_steps["randomforestclassifier"], feature_names=feature_
3 )
```

Out[73]:

Weight	Feature
0.1696 ± 0.0113	fnlwgt
0.1533 ± 0.0396	age
0.1030 ± 0.0348	education
0.0977 ± 0.0479	capital.gain
0.0856 ± 0.0250	hours.per.week
0.0646 ± 0.1385	x1_Married-civ-spouse
0.0489 ± 0.1117	x3_Husband
0.0334 ± 0.0157	capital.loss
0.0286 ± 0.0740	x1_Never-married
0.0205 ± 0.0211	x2_Exec-managerial
0.0193 ± 0.0187	x2_Prof-specialty
0.0118 ± 0.0221	sex
0.0110 ± 0.0225	x3_Wife
0.0094 ± 0.0038	x0_Private
0.0093 ± 0.0242	x3_Not-in-family
0.0080 ± 0.0036	x0_Self-emp-not-inc
0.0078 ± 0.0104	x2_Other-service
0.0066 ± 0.0064	x0_Self-emp-inc
0.0066 ± 0.0239	x3_Own-child
0.0064 ± 0.0024	x4_United-States
... 66 more ...	

Let's compare them with weights what we got with `sklearn feature_importances_`

```
In [74]: 1 data = {
2     "Importance": pipe_rf.named_steps["randomforestclassifier"].feature_
3 }
4 pd.DataFrame(data=data, index=feature_names,).sort_values(
5     by="Importance", ascending=False
6 )[ :10 ]
```

Out[74]:

	Importance
fnlwgt	0.169580
age	0.153339
education	0.102953
capital.gain	0.097686
hours.per.week	0.085583
x1_Married-civ-spouse	0.064646
x3_Husband	0.048896
capital.loss	0.033387
x1_Never-married	0.028629
x2_Exec-managerial	0.020458

- These values tell us globally about which features are important.
- But what if you want to explain a *specific* prediction.
- Some fancier tools can help us do this.

SHAP (SHapley Additive exPlanations)

SHAP (SHapley Additive exPlanations)

- A sophisticated measure of the contribution of each feature.
- [Lundberg and Lee, 2017 \(<https://arxiv.org/pdf/1705.07874.pdf>\)](https://arxiv.org/pdf/1705.07874.pdf)
- We won't go in details. You may refer to [Scott Lundberg's GitHub repo \(<https://github.com/slundberg/shap>\)](https://github.com/slundberg/shap) if you are interested to know more.

General idea

Source

(<https://github.com/slundberg/shap/blob/master/docs/presentations/February%202018%20Talk.ppt>)

General idea

- Provides following kind of explanation
 - Start at a base rate (e.g., how often people get their loans rejected).
 - Add one feature at a time and see how it impacts the decision.

Source

(<https://github.com/slundberg/shap/blob/master/docs/presentations/February%202018%20Talk.ppt>)

Let's try it out on tree-based models.

First you'll have to install it.

```
pip install shap
or
conda install -c conda-forge shap
```

Let's create train and test dataframes with our transformed features.

```
In [75]: 1 X_train_enc = pd.DataFrame(
2     data=preprocessor.transform(X_train).toarray(),
3     columns=feature_names,
4     index=X_train.index,
5 )
6 X_train_enc.head()
```

Out[75]:

	age	fnlwgt	capital.gain	capital.loss	hours.per.week	education	sex	x0_Federal-gov
5514	-0.921955	0.632531	-0.147166	-0.21768	-1.258387	8.0	1.0	0.0
19777	-1.069150	-0.186155	-0.147166	-0.21768	-0.447517	8.0	0.0	0.0
10781	-0.185975	-1.085437	-0.147166	-0.21768	-0.042081	13.0	0.0	0.0
32240	-1.216346	-1.575119	-0.147166	-0.21768	-1.663822	12.0	0.0	0.0
9876	-0.553965	1.588701	-0.147166	-0.21768	-0.042081	13.0	1.0	0.0

5 rows × 86 columns

```
In [76]: 1 X_test_enc = pd.DataFrame(
2     data=preprocessor.transform(X_test).toarray(),
3     columns=feature_names,
4     index=X_test.index,
5 )
6
7 X_test_enc.shape
```

Out[76]: (6513, 86)

Let's get SHAP values for train and test data.

```
In [77]: 1 import shap
2
3 lgbm_explainer = shap.TreeExplainer(pipe_lgbm.named_steps["lgbmclassifier"])
4 train_lgbm_shap_values = lgbm_explainer.shap_values(X_train_enc)
```

LightGBM binary classifier with TreeExplainer shap values output has changed to a list of ndarray

```
In [78]: 1 train_lgbm_shap_values[1].shape
```

Out[78]: (26048, 86)

```
In [79]: 1 test_lgbm_shap_values = lgbm_explainer.shap_values(X_test_enc)
2 test_lgbm_shap_values[1].shape
```

LightGBM binary classifier with TreeExplainer shap values output has changed to a list of ndarray

Out[79]: (6513, 86)

- For classification it's a bit confusing. It gives SHAP arrays both classes.
- Let's stick to shap values for class 1, i.e., income > 50K.

For each example and each feature we have a SHAP value.

In [80]: 1 train_lgbm_shap_values[1]

```
Out[80]: array([[-4.23243013e-01, -5.89878323e-02, -2.65263112e-01, ...,
   9.63030623e-04,  0.00000000e+00,  5.74466631e-04],
  [-6.83190014e-01,  1.15708200e-02, -2.72482485e-01, ...,
   8.17274476e-04,  0.00000000e+00,  8.09406158e-04],
  [ 4.49106369e-01, -1.32455245e-01, -2.39454581e-01, ...,
   8.27603313e-04,  0.00000000e+00,  4.22023416e-03],
  ...,
  [ 1.02714900e+00,  2.38119557e-02, -1.88163464e-01, ...,
   1.13580827e-03,  0.00000000e+00,  6.94390861e-04],
  [ 6.37084418e-01,  2.90573592e-02, -3.03429292e-01, ...,
   9.70726909e-04,  0.00000000e+00,  2.16856964e-03],
  [-1.24950883e+00,  1.19867799e-01, -2.23378846e-01, ...,
   9.70674774e-04,  0.00000000e+00,  9.73838044e-04]])
```

Let's look at the average SHAP values associated with each feature.

In [81]: 1 values = np.abs(train_lgbm_shap_values[1]).mean(0)
2 pd.DataFrame(data=values, index=feature_names, columns=["SHAP"]).sort_v
3 by="SHAP", ascending=False
4)[:10]

	SHAP
x1_Married-civ-spouse	1.086269
age	0.823933
capital.gain	0.572778
education	0.409543
hours.per.week	0.313901
sex	0.188874
capital.loss	0.138607
x3_Own-child	0.112871
x2_Exec-managerial	0.107399
x2_Prof-specialty	0.098181

You can think of this as global feature importances.

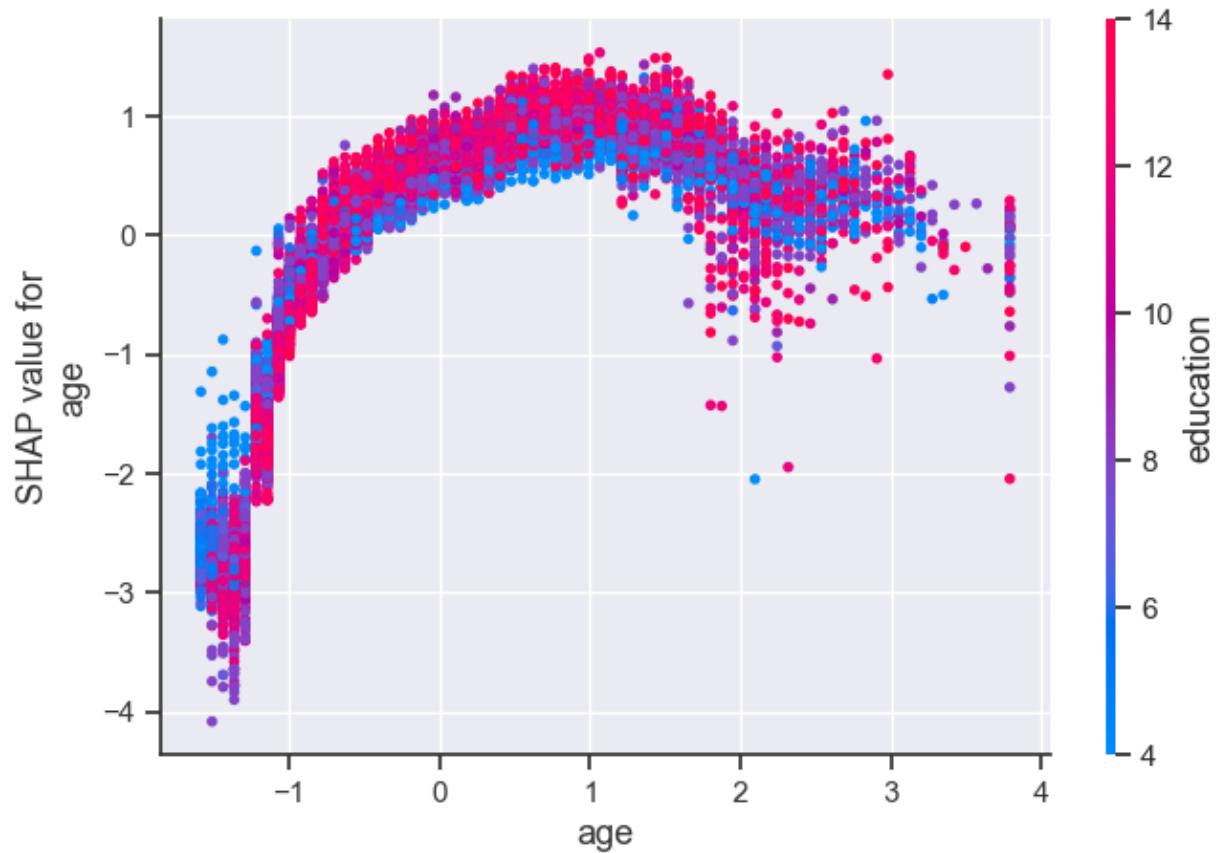
SHAP plots

```
In [82]: 1 # load JS visualization code to notebook
2 shap.initjs()
```



Dependence plot

```
In [83]: 1 shap.dependence_plot("age", train_lgbm_shap_values[1], x_train_enc)
```



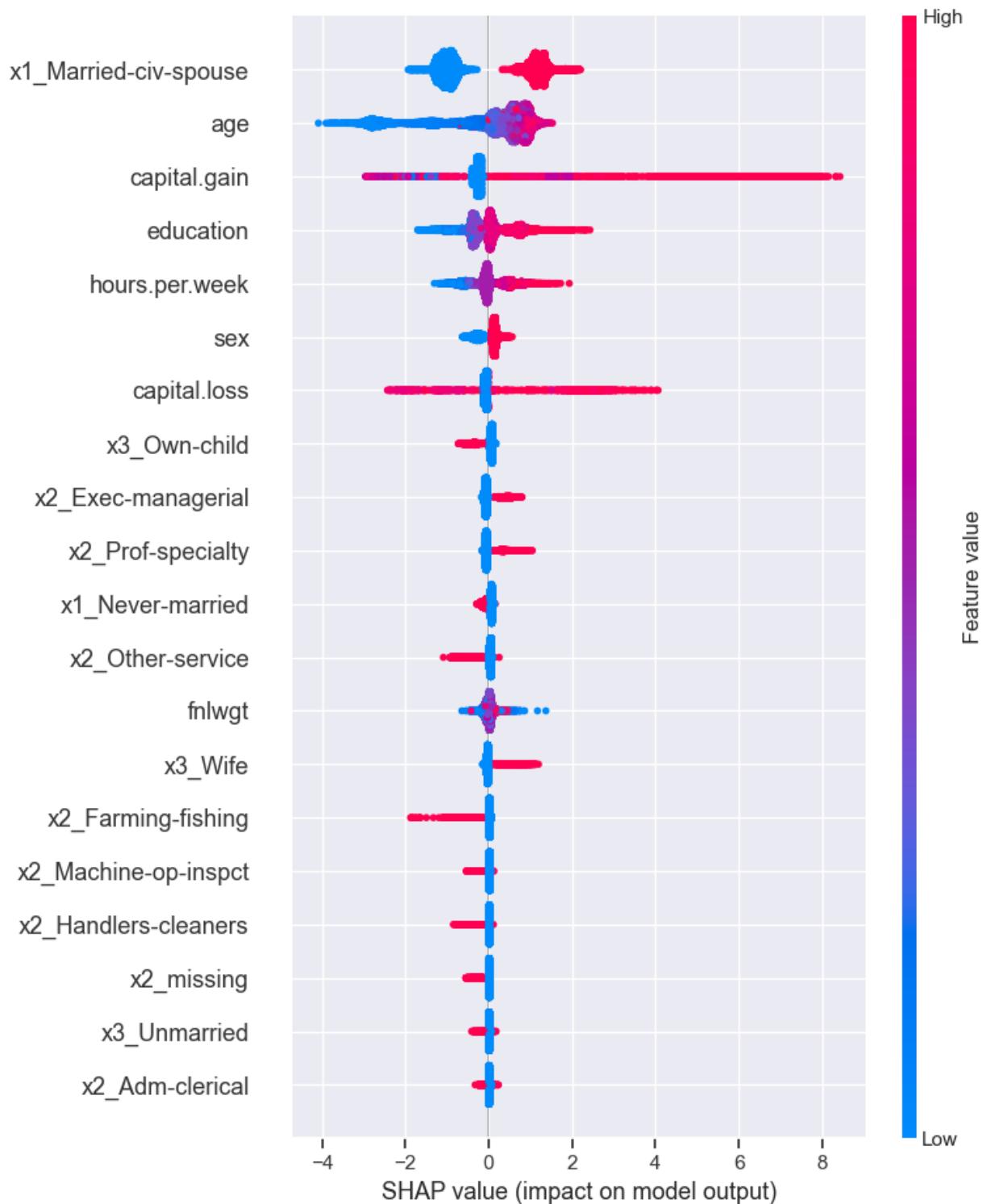
The plot above shows effect of `age` feature on the prediction.

- Each dot is a single prediction for one example.
- The x-axis represents values of the feature `age` (scaled).
- The y-axis is the SHAP value for that feature, which represents how much knowing that feature's value changes the output of the model for that example's prediction.
- Lower values of `age` have smaller SHAP values for class "`>50K`".
- Similarly, higher values of `age` also have a bit smaller SHAP values for class "`>50K`", which makes sense.
- There is some optimal value of `age` between scaled `age` of 1 which gives highest SHAP values for class "`>50K`".

Summary plot

```
In [84]: 1 shap.summary_plot(train_lgbm_shap_values[1], X_train_enc)
```

No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ignored



The plot shows the most important features for predicting the class. It also shows the direction of how it's going to drive the prediction.

- Presence of the marital status of Married-civ-spouse tends to have a large positive SHAP value (makes class 1 more likely), and absence of marital status seems to have negative SHAP values for class 1.
- Higher levels of education seem to have larger SHAP values for class 1, whereas smaller levels of education have smaller SHAP values.

Force plot

- Let's try to explain predictions on a couple of examples from the test data.
- I'm sampling some examples where the target is $\leq 50K$ and some examples where the target is $> 50K$.

```
In [85]: 1 y_test_reset = y_test.reset_index(drop=True)
2 y_test_reset
```

```
Out[85]: 0      <=50K
1      <=50K
2      <=50K
3      <=50K
4      <=50K
...
6508    <=50K
6509    <=50K
6510    >50K
6511    <=50K
6512    >50K
Name: income, Length: 6513, dtype: object
```

```
In [86]: 1 150k_ind = y_test_reset[y_test_reset == "<=50K"].index.tolist()
2 g50k_ind = y_test_reset[y_test_reset == ">50K"].index.tolist()
3
4 ex_150k_index = 150k_ind[10]
5 ex_g50k_index = g50k_ind[10]
```

Example with prediction $\leq 50K$

```
In [87]: 1 pipe_lgbm.named_steps["lgbmclassifier"].predict_proba(X_test_enc)[ex_15]
```

```
Out[87]: array([0.98998011, 0.01001989])
```

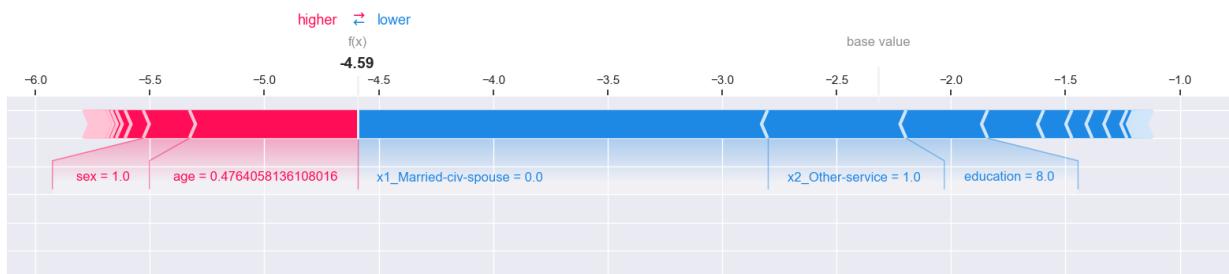
```
In [88]: 1 pipe_lgbm.named_steps["lgbmclassifier"].predict(X_test_enc, raw_score=True)
2     ex_150k_index
3 ] # raw score of the model
```

```
Out[88]: -4.59311253704159
```

```
In [89]: 1 lgbm_explainer.expected_value[1]
```

```
Out[89]: -2.3163172510079377
```

```
In [90]: 1 shap.force_plot(
2     lgbm_explainer.expected_value[1],
3     test_lgbm_shap_values[1][ex_150k_index, :],
4     X_test_enc.iloc[ex_150k_index, :],
5     matplotlib=True,
6 )
```



Example with prediction >50K

```
In [91]: 1 pipe_lgbm.named_steps["lgbmclassifier"].predict_proba(X_test_enc)[ex_g55]
```

```
Out[91]: array([0.47228832, 0.52771168])
```

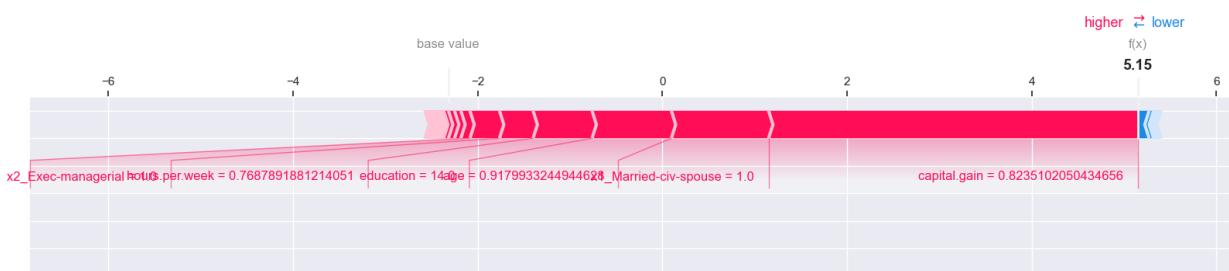
```
In [92]: 1 pipe_lgbm.named_steps["lgbmclassifier"].predict(X_test_enc, raw_score=True)
2 ex_g50k_index
3 ] # raw model score
```

```
Out[92]: 0.11096043410156158
```

```
In [93]: 1 g50k_ind[:10]
```

```
Out[93]: [17, 18, 30, 31, 39, 45, 49, 58, 59, 62]
```

```
In [94]: 1 shap.force_plot(
2     lgbm_explainer.expected_value[1],
3     test_lgbm_shap_values[1][18, :],
4     X_test_enc.iloc[18, :],
5     matplotlib=True,
6 )
```



```
In [95]: 1 test_lgbm_shap_values[1][ex_g50k_index, :],
```

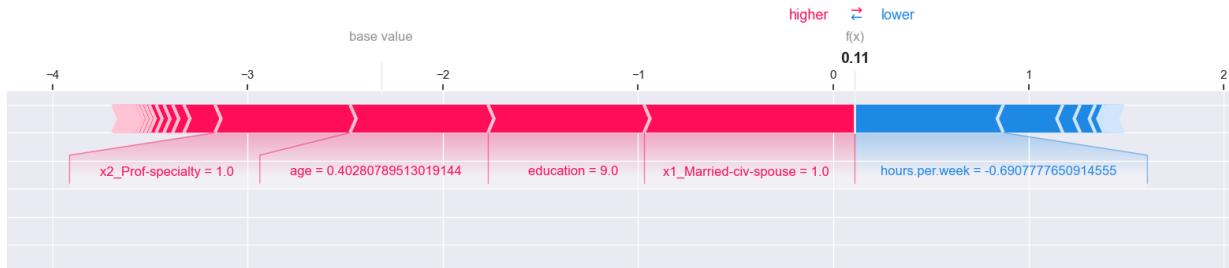
```
Out[95]: (array([ 7.06576926e-01,  4.48050353e-02, -3.04826888e-01, -8.90345564e-02,
       -7.60234999e-01,  7.99009432e-01,  1.59145115e-01, -1.04561236e-02,
       1.12544519e-02,  0.00000000e+00, -7.93711497e-03, -8.36423421e-03,
      -4.36341788e-02,  7.59578850e-03,  0.00000000e+00,  2.58025291e-03,
     -1.21233828e-03,  0.00000000e+00,  1.08073198e+00, -1.96300214e-03,
     4.05063596e-02,  3.89528728e-03, -3.40493462e-03,  1.13966166e-02,
     0.00000000e+00,  1.78540517e-02, -9.45890343e-03,  2.07052478e-02,
     4.68861970e-03,  2.04262116e-02,  5.73720200e-02,  4.04716758e-03,
     6.91720232e-01, -4.07221602e-03, -1.66621103e-02, -1.42348633e-02,
     8.08173720e-03,  1.52914770e-02,  8.16100692e-03, -1.52031143e-02,
     4.30523314e-03,  3.49488337e-02,  1.95922173e-02, -7.73110056e-02,
     0.00000000e+00, -1.86454897e-04,  9.32582520e-04,  2.43918385e-03,
    -3.25037985e-04,  1.48300302e-03,  0.00000000e+00,  0.00000000e+00,
     0.00000000e+00,  0.00000000e+00, -2.05106608e-04,  2.38263534e-04,
     0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
     0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
     0.00000000e+00, -9.45855608e-04,  0.00000000e+00,  0.00000000e+00,
     0.00000000e+00,  7.16936676e-03,  0.00000000e+00,  0.00000000e+00,
     0.00000000e+00, -2.83890851e-03,  0.00000000e+00,  0.00000000e+00,
     2.26468523e-03,  0.00000000e+00,  2.02526698e-04,  0.00000000e+00,
     0.00000000e+00,  0.00000000e+00,  9.58695151e-03,  9.66566029e-04,
     0.00000000e+00, -1.84802303e-04]),)
```

In [96]:

```

1 shap.force_plot(
2     lgbm_explainer.expected_value[1],
3     test_lgbm_shap_values[1][ex_g50k_index, :],
4     X_test_enc.iloc[ex_g50k_index, :],
5     matplotlib=True,
6 )

```



Observations:

- Everything is with respect to class 1 here.
- The base value for class 1 is -2.316. (You can think of this as the average raw score.)
- We see the forces that drive the prediction.
- That is, we can see the main factors pushing it from the base value (average over the dataset) to this particular prediction.
- Features that push the prediction to a higher value are shown in red.
- Features that push the prediction to a lower value are shown in blue.

Note: a nice thing about SHAP values is that the feature importances sum to the prediction:

In [97]: 1 test_lgbm_shap_values[1][ex_g50k_index, :].sum() + lgbm_explainer.expec

Out[97]: 0.11096043410156309

Provides explainer for different kinds of models

- [TreeExplainer](https://shap.readthedocs.io/en/latest/) (<https://shap.readthedocs.io/en/latest/>) (supports XGBoost, CatBoost, LightGBM)
- [DeepExplainer](https://shap.readthedocs.io/en/latest/index.html#shap.DeepExplainer) (<https://shap.readthedocs.io/en/latest/index.html#shap.DeepExplainer>) (supports deep-learning models)
- [KernelExplainer](https://shap.readthedocs.io/en/latest/index.html#shap.KernelExplainer) (<https://shap.readthedocs.io/en/latest/index.html#shap.KernelExplainer>) (supports kernel-based models)
- [GradientExplainer](https://shap.readthedocs.io/en/latest/index.html#shap.GradientExplainer) (<https://shap.readthedocs.io/en/latest/index.html#shap.GradientExplainer>) (supports Keras and Tensorflow models)

- Can also be used to explain text classification and image classification
- Example: In the picture below, red pixels represent positive SHAP values that increase the probability of the class, while blue pixels represent negative SHAP values that reduce the probability of the class.

Source (<https://github.com/slundberg/shap>)

Other tools

- [lime](https://github.com/marcotcr/lime) (<https://github.com/marcotcr/lime>) is another package.

If you're not already impressed, keep in mind:

- So far we've only used sklearn models.
- Most sklearn models have some built-in measure of feature importances.
- On many tasks we need to move beyond sklearn, e.g. LightGBM, deep learning.
- These tools work on other models as well, which makes them extremely useful.

Why do we want this information?

Possible reasons:

- Identify features that are not useful and maybe remove them.
- Get guidance on what new data to collect.
 - New features related to useful features -> better results.
 - Don't bother collecting useless features -> save resources.
- Help explain why the model is making certain predictions.
 - Debugging, if the model is behaving strangely.
 - Regulatory requirements.
 - Fairness / bias.
 - Keep in mind this can be used on **deployment** predictions!

? ? Questions for you

True/False

1. You train a random forest on a binary classification problem with two classes [neg, pos]. A value of 0.580 for feat1 given by `feature_importances_` attribute of your model means that increasing the value of feat1 will drive us towards positive class.
2. eli5 can be used to get feature importances for non `sklearn` models.
3. With SHAP you can only explain predictions on the training examples.

In []:

1

CPSC 330

Applied Machine Learning

Lecture 13: Feature engineering and feature selection

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

```
In [1]: 1 import os
2 import sys
3
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import numpy.random as npr
7 import pandas as pd
8 from sklearn.compose import (
9     ColumnTransformer,
10    TransformedTargetRegressor,
11    make_column_transformer,
12 )
13 from sklearn.dummy import DummyRegressor
14 from sklearn.ensemble import RandomForestRegressor
15 from sklearn.impute import SimpleImputer
16 from sklearn.linear_model import LinearRegression, LogisticRegression,
17 from sklearn.metrics import make_scorer, mean_squared_error, r2_score
18 from sklearn.model_selection import cross_val_score, cross_validate, train_test_split
19 from sklearn.pipeline import Pipeline, make_pipeline
20 from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
21 from sklearn.svm import SVC
22 from sklearn.tree import DecisionTreeRegressor
```

```
In [3]: 1 import sys
2 sys.path.append("../code/.")
```

Learning outcomes

From this lecture, students are expected to be able to:

- Explain what feature engineering is and the importance of feature engineering in building machine learning models.

- Carry out preliminary feature engineering on text data.
- Explain the general concept of feature selection.
- Discuss and compare different feature selection methods at a high level.
- Use `sklearn`'s implementation of recursive feature elimination (`RFE`) and forward and backward selection (`SequentialFeatureSelector`).

Feature engineering: Motivation

What is feature engineering?

- Better features: more flexibility, higher score, we can get by with simple and more interpretable models.
- If your features, i.e., representation is bad, whatever fancier model you build is not going to help ("garbage in, garbage out"). This is less true now with deep learning.

Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data.

- Jason Brownlee

Some quotes on feature engineering

A quote by Pedro Domingos [A Few Useful Things to Know About Machine Learning](https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf) (<https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf>)

... At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used.

A quote by Andrew Ng, [Machine Learning and AI via Brain simulations](https://ai.stanford.edu/~ang/slides/DeepLearning-Mar2013.pptx) (<https://ai.stanford.edu/~ang/slides/DeepLearning-Mar2013.pptx>)

Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.

Better features usually help more than a better model.

- Good features would ideally:
 - capture most important aspects of the problem
 - allow learning with few examples
 - generalize to new scenarios.
- There is a trade-off between simple and expressive features:
 - With simple features overfitting risk is low, but scores might be low.

- With complicated features scores can be high, but so is overfitting risk.

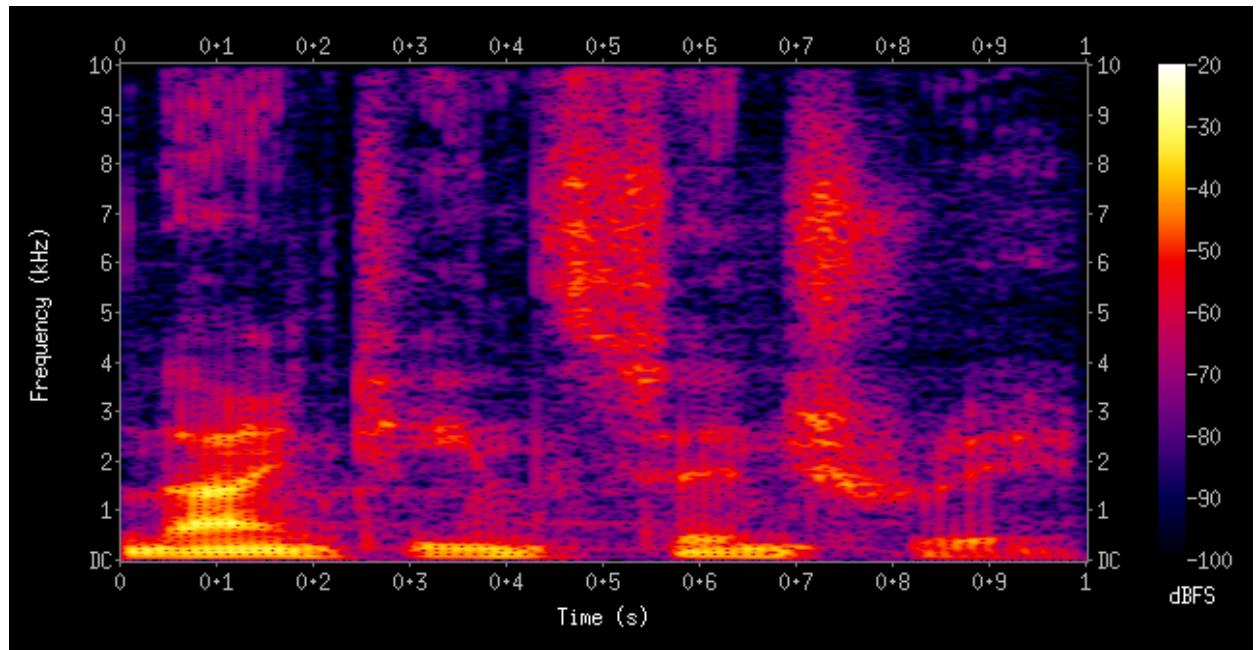
The best features may be dependent on the model you use.

- Examples:
 - For counting-based methods like decision trees separate relevant groups of variable values
 - Discretization makes sense
 - For distance-based methods like KNN, we want different class labels to be "far".
 - Standardization
 - For regression-based methods like linear regression, we want targets to have a linear dependency on features.

Domain-specific transformations

In some domains there are natural transformations to do:

- Spectrograms (sound data)
- Wavelets (image data)
- Convolutions



[Source \(<https://en.wikipedia.org/wiki/Spectrogram>\)](https://en.wikipedia.org/wiki/Spectrogram)

In this lecture, I'll show you two example domains where feature engineering plays an important role:

- Text data
- Audio data

Common features used in text classification

Bag of words

- So far for text data we have been using bag of word features.
- They are good enough for many tasks. But ...
- This encoding throws out a lot of things we know about language
- It assumes that word order is not that important.
- So if you want to improve the scores further on text classification tasks you carry out **feature engineering**.

Let's look at some examples from research papers.

Example: Label "Personalized" Important E-mails:

- [The Learning Behind Gmail Priority Inbox](https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/36955.pdf)
[\(https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/36955.pdf\)](https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/36955.pdf)
- Features: bag of words, trigrams, regular expressions, and so on.
- There might be some "globally" important messages:
 - "This is your mother, something terrible happened, give me a call ASAP."
- But your "important" message may be unimportant to others.
 - Similar for spam: "spam" for one user could be "not spam" for another.

- Social features (e.g., percentage of sender emails that is read by the recipient)
- Content features (e.g., recent terms the user has been using in emails)
- Thread features (e.g., whether the user has started the thread)
- ...

[The Learning Behind Gmail Priority Inbox](https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/36955.pdf)

<https://static.googleusercontent.com/media/research.google.com/en//pi>

2.1 Features

Feature engineering examples: [Automatically Identifying Good Conversations Online \(http://www.courtneynapoles.com/res/icwsm17-automatically.pdf\)](http://www.courtneynapoles.com/res/icwsm17-automatically.pdf).

BOW (21k)	Counts of tokens.
Embeddings (300)	Averaged word embedding values from Google News vectors (Mikolov et al. 2013).
Entity (12)	Counts of named entity types.
Length (2)	Mean # sentences/comment, # tokens/sentence.
Lexicon (6)	# pronouns; agreement and certainty phrases; discourse connectives; and abusive language.
POS (23k)	Counts of 1–3-gram POS tags.
Popularity (4)	# thumbs up (TU), # thumbs down (TD), TU + TD, and $\frac{TU}{TU+TD}$.
Similarity (8)	Overlap between comment and headline, first comment, previous comment, and all previous comments (if applicable).
User (7)	# comments posted, # threads participated in, # threads initiated, TU and TD received, and commenting rate.

Table 4: Features used in the linear model. The number of features from each group is indicated in parentheses.

(optional) Term weighing (TF-IDF)

- A measure of relatedness between words and documents
- Intuition: Meaningful words may occur repeatedly in related documents, but functional words (e.g., *make*, *the*) may be distributed evenly over all documents

$$\text{tf.idf}(w_i, d_j) = (1 + \log(\text{tf}_{ij})) \log \frac{D}{\text{df}_i}$$

where,

- $\text{tf}_{ij} \rightarrow$ (term frequency) number of occurrences of the term w_i in document d_j
- $D \rightarrow$ number of documents
- $\text{df}_i \rightarrow$ (document frequency) number of documents in which w_i occurs

Check TfidfVectorizer from sklearn .

N-grams

- Incorporating more context
- A contiguous sequence of n items (characters, tokens) in text.

CPSC330 students are hard-working .

- 2-grams (bigrams): a contiguous sequence of two words
 - CPSC330 students, students are, are hard-working .
- 3-grams (trigrams): a contiguous sequence of three words
 - CPSC330 students are, students are hard-working .

You can extract ngram features using CountVectorizer by passing ngram_range .

```
In [2]: 1 from sklearn.feature_extraction.text import CountVectorizer
2
3 X = [
4     "URGENT!! As a valued network customer you have been selected to re
5     "Lol you are always so convincing.",
6     "URGENT!! Call right away!!",
7 ]
8 vec = CountVectorizer(ngram_range=(1, 3))
9 X_counts = vec.fit_transform(X)
10 bow_df = pd.DataFrame(X_counts.toarray(), columns=vec.get_feature_names)
```

In [3]: 1 bow_df

	900 prize	900 reward	900 prize	always	always so	always so convincing	are	are always	are always	are so	as	urgent	ui
URGENT!! <i>As a valued network customer you have been selected to receive a \$900 prize reward!</i>	1	1	1	0	0	0	0	0	0	0	1	...	0	0
Lol you are always so convincing.	0	0	0	1	1	1	1	1	1	1	0	...	0	0
URGENT!! <i>Call right away!!</i>	0	0	0	0	0	0	0	0	0	0	0	...	1	1

3 rows × 61 columns

ASIDE: [Google n-gram viewer \(https://books.google.com/ngrams\)](https://books.google.com/ngrams)

- All Our N-gram are Belong to You
 - [\(https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-toyou.html\)](https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-toyou.html)

Here at Google Research we have been using word n-gram models for a variety of R&D projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times."

In [4]:

```

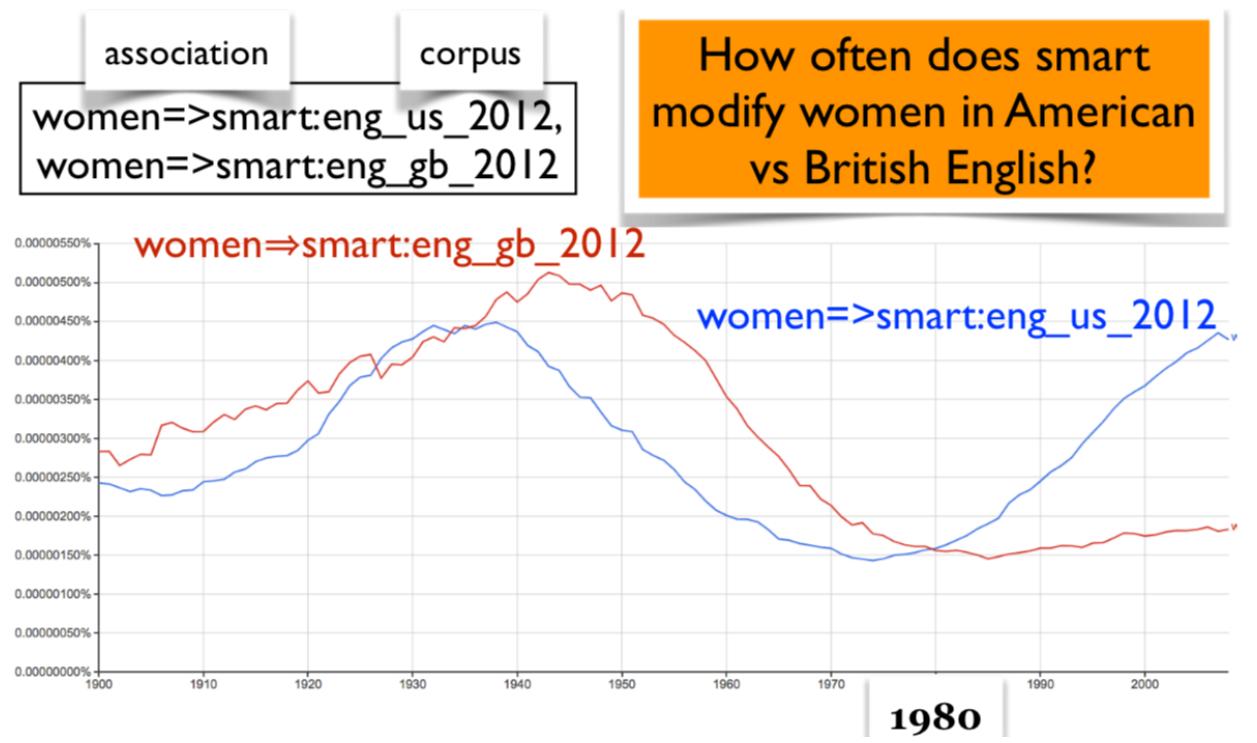
1 import IPython
2
3 url = "https://books.google.com/ngrams/"
4 IPython.display.IFrame(url, width=1000, height=800)

```

Out[4]:

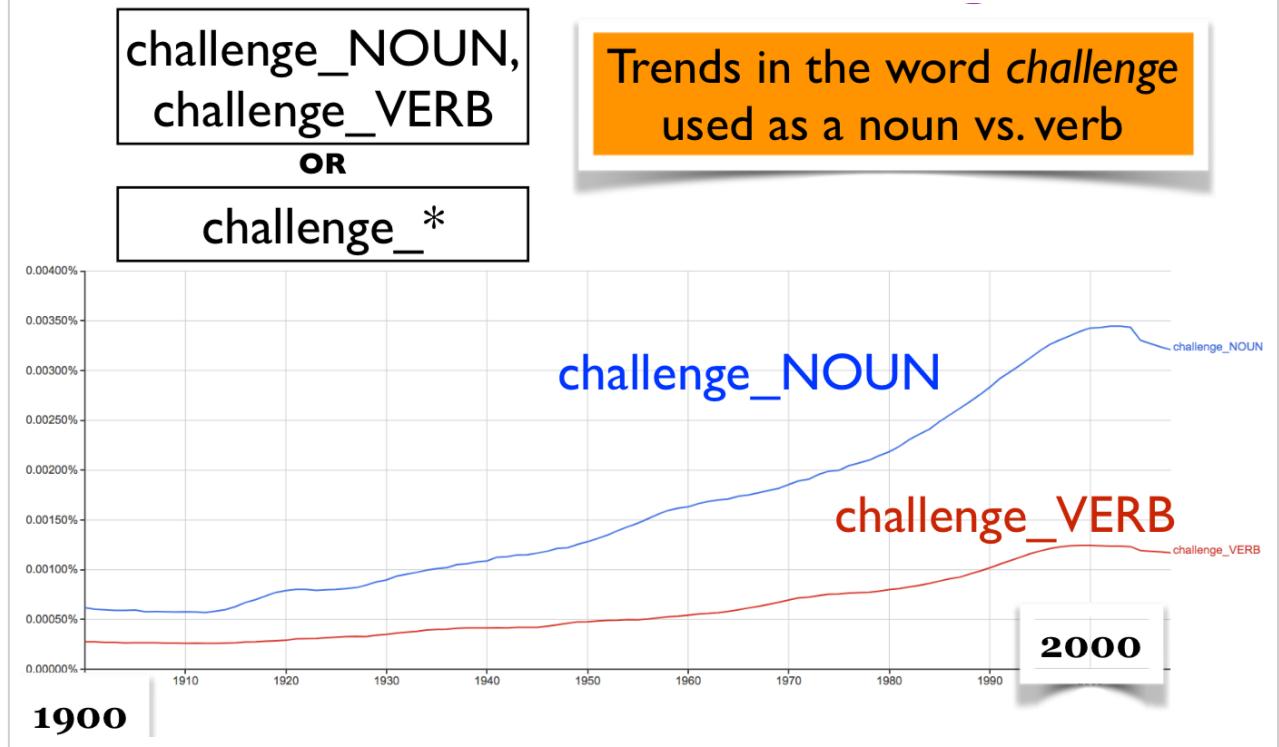
Aside: [Google n-gram viewer \(https://books.google.com/ngrams\)](https://books.google.com/ngrams)

- Count the occurrences of the bigram *smart women* in the corpus from 1900 to 2000



Aside: [Google n-gram viewer \(<https://books.google.com/ngrams>\)](https://books.google.com/ngrams)

- Trends in the word *challenge* used as a noun vs. verb



Part-of-speech features

Part-of-speech (POS) in English

- Part-of-speech: A kind of syntactic category that tells you some of the grammatical properties of a word.
 - Noun → water, sun, cat
 - Verb → run, eat, teach

The ___ was running.

- Only a noun fits here.

Part-of-speech (POS) features

- POS features use POS information for the words in text.

CPSC330/PROPER_NOUN students/NOUN are/VERB hard-working/ADJECTIVE

An example from a project

- Data: a bunch of documents
- Task: identify texts with *permissions* and identify who is giving permission to whom.

You may **disclose** Google confidential information when compelled to do so by law if **you** provide **us** reasonable prior notice, unless a court orders that **we** not receive notice.

- A very simple solution
 - Look for pronouns and verbs.
 - Add POS tags as features in your model.
 - Maybe look up words similar to **disclose**.

Penn Treebank part-of-speech tags (bonus)

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	<i>and, but, or</i>	PDT	predeterminer	<i>all, both</i>	VBP	verb non-3sg present	<i>eat</i>
CD	cardinal number	<i>one, two</i>	POS	possessive ending	's	VBZ	verb 3sg pres	<i>eats</i>
DT	determiner	<i>a, the</i>	PRP	personal pronoun	<i>I, you, he</i>	WDT	wh-determ.	<i>which, that</i>
EX	existential 'there'	<i>there</i>	PRP\$	possess. pronoun	<i>your, one's</i>	WP	wh-pronoun	<i>what, who</i>
FW	foreign word	<i>mea culpa</i>	RB	adverb	<i>quickly</i>	WP\$	wh-possess.	<i>whose</i>
IN	preposition/ subordin-conj	<i>of, in, by</i>	RBR	comparative	<i>faster</i>	WRB	wh-adverb	<i>how, where</i>
JJ	adjective	<i>yellow</i>	RBS	superlatv. adverb	<i>fastest</i>	\$	dollar sign	\$
JJR	comparative adj	<i>bigger</i>	RP	particle	<i>up, off</i>	#	pound sign	#
JJS	superlative adj	<i>wildest</i>	SYM	symbol	<i>+, %, &</i>	"	left quote	' or "
LS	list item marker	<i>1, 2, One</i>	TO	"to"	<i>to</i>	"	right quote	' or "
MD	modal	<i>can, should</i>	UH	interjection	<i>ah, oops</i>	(left paren	[, (, {, <
NN	sing or mass noun	<i>llama</i>	VB	verb base form	<i>eat</i>)	right paren],), }, >
NNS	noun, plural	<i>llamas</i>	VBD	verb past tense	<i>ate</i>	,	comma	,
NNP	proper noun, sing.	<i>IBM</i>	VBG	verb gerund	<i>eating</i>	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	<i>Carolinas</i>	VBN	verb past part.	<i>eaten</i>	:	sent-mid punc	: ; ... --

- How do we extract part-of-speech information?
- We use **pre-trained models!**

- A couple of popular libraries which include such pre-trained models.

- `nltk`

```
conda install -c anaconda nltk
```

- `spaCy`

```
conda install -c conda-forge spacy
```

In [5]:

```

1 import nltk
2
3 nltk.download("punkt")

```

[nltk_data] Downloading package punkt to /Users/mathias/nltk_data...
[nltk_data] Package punkt is already up-to-date!

Out[5]: True

- You also need to download the language model which contains all the pre-trained models. For that run the following in your course `conda` environment.

In [6]:

```
1 # python -m spacy download en_core_web_md
```

[spaCy \(<https://spacy.io/>\)](https://spacy.io/)

A useful package for text processing and feature extraction

- Active development: <https://github.com/explosion/spaCy> (<https://github.com/explosion/spaCy>)
- Interactive lessons by Ines Montani: <https://course.spacy.io/en/> (<https://course.spacy.io/en/>)
- Good documentation, easy to use, and customizable.

In [7]:

```

1 import en_core_web_md # pre-trained model
2 import spacy
3
4 nlp = en_core_web_md.load()

```

In [8]:

```

1 sample_text = """Dolly Parton is a gift to us all.
2 From writing all-time great songs like "Jolene" and "I Will Always Love
3 to great performances in films like 9 to 5, to helping fund a COVID-19
4 she's given us so much. Now, Netflix bring us Dolly Parton's Christmas
5 an original musical that stars Christine Baranski as a Scrooge-like lan
6 who threatens to evict an entire town on Christmas Eve to make room for
7 Directed and choreographed by the legendary Debbie Allen and counting J
8 and Parton herself amongst its cast, Christmas on the Square seems like
9 to save Christmas 2020. 😊 🙌 """
10
11 # [Adapted from here.](https://thepopbreak.com/2020/11/22/dolly-partons)

```

Spacy extracts all interesting information from text with this call.

In [9]:

```
1 doc = nlp(sample_text)
```

Let's look at part-of-speech tags.

```
In [10]: 1 print([(token, token.pos_) for token in doc][:20])
```

```
[(Dolly, 'PROPN'), (Parton, 'PROPN'), (is, 'AUX'), (a, 'DET'), (gift, 'NO UN'), (to, 'ADP'), (us, 'PRON'), (all, 'PRON'), (., 'PUNCT'), ('SPACE'), (From, 'ADP'), (writing, 'VERB'), (all, 'DET'), (-, 'PUNCT'), (time, 'NOUN'), (great, 'ADJ'), (songs, 'NOUN'), (like, 'ADP'), (", 'PUNC T'), (Jolene, 'PROPN')]
```

- Often we want to know who did what to whom.
- **Named entities** give you this information.
- What are named entities in the text?

```
In [11]: 1 print("Named entities:\n", [(ent.text, ent.label_) for ent in doc.ents])
2 print("\nORG means: ", spacy.explain("ORG"))
3 print("\nPERSON means: ", spacy.explain("PERSON"))
4 print("\nDATE means: ", spacy.explain("DATE"))
```

Named entities:

```
[('Dolly Parton', 'PERSON'), ('Jolene', 'PERSON'), ('9 to 5', 'DATE'),
('Netflix', 'ORG'), ('Dolly Parton', 'PERSON'), ('Christmas', 'DATE'),
('Square', 'FAC'), ('Christine Baranski', 'PERSON'), ('Christmas Eve', 'DATE'),
('Debbie Allen', 'PERSON'), ('Jennifer Lewis', 'PERSON'), ('Parton',
'PERSON'), ('Christmas', 'DATE'), ('Square', 'FAC'), ('Christmas 202
0', 'DATE')]
```

ORG means: Companies, agencies, institutions, etc.

PERSON means: People, including fictional

DATE means: Absolute or relative dates or periods

In [12]:

```

1 from spacy import displacy
2
3 displacy.render(doc, style="ent")

```

Dolly Parton PERSON is a gift to us all.

From writing all-time great songs like " Jolene PERSON " and "I Will Always Love You", to great performances in films like 9 to 5 DATE , to helping fund a COVID-19 vaccine, she's given us so much. Now, Netflix ORG bring us Dolly Parton PERSON 's Christmas DATE on the Square FAC , an original musical that stars Christine Baranski PERSON as a Scrooge-like landowner who threatens to evict an entire town on Christmas Eve DATE to make room for a new mall. Directed and choreographed by the legendary Debbie Allen PERSON and counting Jennifer Lewis PERSON and Parton PERSON herself amongst its cast, Christmas DATE on the Square FAC seems like the perfect movie to save Christmas 2020 DATE . 😊 🙌

An example from a project

Goal: Extract and visualize inter-corporate relationships from disclosed annual 10-K reports of public companies.

[Source for the text below. \(https://www.bbc.com/news/business-39875417\)](https://www.bbc.com/news/business-39875417)

In [13]:

```

1 text = (
2     "Heavy hitters, including Microsoft and Google, "
3     "are competing for customers in cloud services with the likes of IB
4 )

```

In [14]:

```

1 doc = nlp(text)
2 displacy.render(doc, style="ent")
3 print("Named entities:\n", [(ent.text, ent.label_) for ent in doc.ents])

```

Heavy hitters, including Microsoft ORG and Google ORG , are competing for customers in cloud services with the likes of IBM ORG and Salesforce PRODUCT .

Named entities:

```
[('Microsoft', 'ORG'), ('Google', 'ORG'), ('IBM', 'ORG'), ('Salesforce', 'PRODUCT')]
```

If you want emoji identification support install `spacymoji` (<https://pypi.org/project/spacymoji/>) in the course environment.

```
pip install spacymoji
```

After installing `spacymoji`, if it's still complaining about module not found, my guess is that you do not have `pip` installed in your `conda` environment. Go to your course `conda` environment install `pip` and install the `spacymoji` package in the environment using the `pip` you just installed in the current environment.

```
conda install pip
YOUR_MINICONDA_PATH/miniconda3/envs/cpsc330/bin/pip install spacymoji
```

In [15]:

```
1 from spacymoji import Emoji
2
3 nlp.add_pipe("emoji", first=True);
```

Does the text have any emojis? If yes, extract the description.

In [16]:

```
1 doc = nlp(sample_text)
2 doc._.emoji
```

Out[16]:

```
[('😊', 138, 'smiling cat face with heart-eyes'),
 ('👍', 139, 'thumbs up dark skin tone')]
```

Final remarks

- If we want to go beyond bag-of-words and incorporate human knowledge in models, we carry out feature engineering.
- Some common features include:
 - ngram features
 - part-of-speech features
 - named entity features
 - emoticons in text
- These are usually extracted from pre-trained models using libraries such as `spaCy`.
- Now a lot of this has moved to deep learning.
- But industries still rely on manual feature engineering.

Classify music style from audio files

- Imagine you are asked to develop a music style classification system to help a song recommendation system.

- E.g., something similar to what Spotify does
- Training data: audio files along with their music styles (e.g., classical, blues, pop)
- Prediction task: Given a new raw audio file predict the music style of in the audio.
- You can download the data from [MARSYAS](http://marsyas.info/downloads/datasets.html) (<http://marsyas.info/downloads/datasets.html>). You can also get it [from kaggle](https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification) (<https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification>) in the genres_original folder.
 - Beware of the size; it is 1.2 GB.

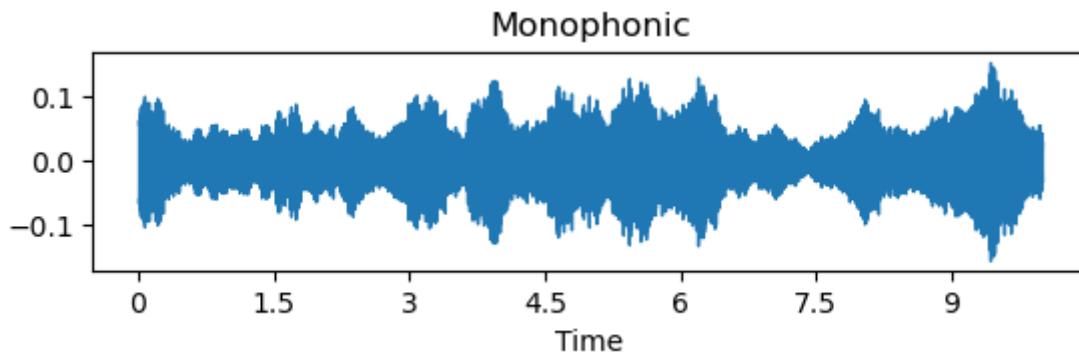
- We'll be using librosa library for feature engineering of audio files.
- You can install librosa in your conda environment as follows:

```
conda install -c conda-forge librosa
```

In [17]:

```
1 import librosa.display
2 import matplotlib.pyplot as plt
3
4 y, sr = librosa.load(
5     "../data/genres/classical/classical.00001.wav", duration=10
6 )
7 plt.figure()
8 plt.subplot(3, 1, 1)
9 librosa.display.waveform(y, sr=sr)
10 plt.title("Monophonic")
```

Out[17]: Text(0.5, 1.0, 'Monophonic')



In [18]:

```
1 import IPython.display
2
3 IPython.display.Audio(y, rate=sr)
```

Out[18]:

0:00 / 0:00

- I have taken a subset of the above dataset with genres and stored it in music_genres_small
 - blues
 - classical
 - pop

- rock
- Let's extract some domain-specific features called [MFCC features](#) (https://en.wikipedia.org/wiki/Mel-frequency_cepstrum) from the above subset and store it as a CSV. I am pushing this CSV in the repo.
- (You do not need to understand the feature extraction code below.)

In [19]:

```
1 data = {}  
2 import glob  
3 import ntpath
```

In [20]:

```

1 data_dir = "../data/music_genres_small/"
2 raw_audio_X = np.empty(shape=(400, 1))
3 raw_audio_X10 = np.empty(shape=(400, 10))
4 raw_audio_X10k = np.empty(shape=(400, 10000))
5 raw_audio_y = []
6 row_ind = 0
7
8 for audio_file in glob.glob(data_dir + r"/*/*.wav"):
9     y, sr = librosa.load(audio_file, mono=True, duration=30)
10    mfcc = librosa.feature.mfcc(y=y, sr=sr)
11    chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
12    spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
13    spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
14    rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
15    zcr = librosa.feature.zero_crossing_rate(y)
16    filename = ntpath.basename(audio_file)
17    label = filename.split(".")[0]
18
19    data.setdefault("audio_file", []).append(filename)
20    data.setdefault("label", []).append(label)
21    data.setdefault("chroma_stft", []).append(np.mean(chroma_stft))
22    data.setdefault("spec_cent", []).append(np.mean(spec_cent))
23    data.setdefault("spec_bw", []).append(np.mean(spec_bw))
24    data.setdefault("rolloff", []).append(np.mean(rolloff))
25    data.setdefault("zcr", []).append(np.mean(zcr))
26
27    # Get mfcc features
28    feat_prefix = "mfcc"
29    ind = 1
30    for feat in mfcc:
31        key = feat_prefix + str(ind)
32        data.setdefault(key, []).append(np.mean(feat))
33        ind += 1
34    try:
35        raw_audio_X[row_ind] = np.mean(y)
36        raw_audio_X10[row_ind] = y[:10]
37        raw_audio_X10k[row_ind] = y[:10000]
38        raw_audio_y.append(label)
39    except:
40        print("Broadcasting problem with file %s" % (audio_file))
41        row_ind += 1
42
43 df = pd.DataFrame(data)
44 df.to_csv("../data/genres/music_genre.csv", index=False)

```

If we just pass raw audio data to an ML model.

- We could just take the mean of the audio time series which won't be much meaningful in each case.

In [21]:

```

1 raw_audio_y = np.asarray(raw_audio_y)
2 raw_audio_y.shape
3 X_train, X_test, y_train, y_test = train_test_split(
4     raw_audio_X, raw_audio_y, test_size=0.20, random_state=111
5 )
6 lr = LogisticRegression(solver="liblinear")
7 pd.DataFrame(cross_validate(lr, X_train, y_train, return_train_score=True))

```

Out[21]:

	fit_time	score_time	test_score	train_score
0	0.000849	0.000189	0.250000	0.261719
1	0.000447	0.000114	0.250000	0.261719
2	0.000440	0.000105	0.265625	0.257812
3	0.000416	0.000105	0.265625	0.257812
4	0.000408	0.000105	0.265625	0.257812

If we just pass raw audio data to an ML model.

- We could just take the mean of the audio time series which won't be much meaningful in each case...
- Or we could take the beginning of the series... This is better but still not great at all.

In [22]:

```

1 raw_audio_y = np.asarray(raw_audio_y)
2 results = {}
3 for X in [raw_audio_X10, raw_audio_X10k]:
4     X_train, X_test, y_train, y_test = train_test_split(
5         X, raw_audio_y, test_size=0.20, random_state=111
6     )
7     lr = LogisticRegression(solver="liblinear")
8     res = pd.DataFrame(cross_validate(lr, X_train, y_train, return_train=True))
9     print(res)
10    print(np.mean(res['test_score']))
11

```

	fit_time	score_time	test_score	train_score
0	0.001102	0.000164	0.312500	0.398438
1	0.000793	0.000112	0.312500	0.371094
2	0.000748	0.000105	0.343750	0.359375
3	0.000740	0.000103	0.296875	0.351562
4	0.000755	0.000105	0.375000	0.386719
0.328125				
	fit_time	score_time	test_score	train_score
0	1.450364	0.000750	0.312500	1.0
1	1.323437	0.000709	0.390625	1.0
2	1.368735	0.000692	0.328125	1.0
3	1.395656	0.000719	0.359375	1.0
4	1.375528	0.000713	0.312500	1.0
0.340625				

Let's try it with standard domain-specific MFCC features.

In [23]:

```

1 music_csv_df = pd.read_csv("../data/genres/music_genre.csv")
2 music_csv_df.columns
3 X = music_csv_df.drop(columns=["audio_file", "label"])
4 y = music_csv_df["label"]
5 X_train, X_test, y_train, y_test = train_test_split(
6     X, y, test_size=0.20, random_state=111
7 )
8 lr = LogisticRegression(solver="liblinear")
9 pd.DataFrame(cross_validate(lr, X_train, y_train, return_train_score=True))

```

Out[23]:

	fit_time	score_time	test_score	train_score
0	0.008478	0.000660	0.703125	0.914062
1	0.008674	0.000644	0.796875	0.906250
2	0.008776	0.000586	0.781250	0.898438
3	0.007765	0.000571	0.765625	0.890625
4	0.008103	0.000565	0.781250	0.898438

- Much better results with domain-specific features!!
- You could improve it further with more careful feature engineering.

Summary

- Feature engineering is finding the useful representation of the data that can help us effectively solve our problem.
- We looked at commonly used features in text data
 - Bag of word features
 - N-gram features
 - Part-of-speech features
 - Classify music style from audio files

Feature engineering

- The best features are application-dependent.
- It's hard to give general advice. But here are some guidelines.
 - Ask the domain experts.
 - Go through academic papers in the discipline.
 - Often have idea of right discretization/standardization/transformation.
 - If no domain expert, cross-validation will help.
- If you have lots of data, use deep learning methods.

The algorithms we used are very standard for Kagglers ... We spent most of our efforts in feature engineering...
 - Xavier Conort, on winning the Flight Quest challenge on Kaggle

Break (5 min)



Feature selection: Introduction and motivation

- With so many ways to add new features, we can increase dimensionality of the data.
- More features means more complex models, which means increasing the chance of overfitting.

What is feature selection?

- Find the features (columns) X that are important for predicting y , and remove the features that aren't.
- Given $X = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}$ and $y = \begin{bmatrix} \end{bmatrix}$, find the columns $1 \leq j \leq n$ in X that are important for predicting y .

Why feature selection?

- Interpretability: Models are more interpretable with fewer features. If you get the same performance with 10 features instead of 500 features, why not use the model with smaller number of features?
- Computation: Models fit/predict faster with fewer columns.
- Data collection: What type of new data should I collect? It may be cheaper to collect fewer columns.
- Fundamental tradeoff: Can I reduce overfitting by removing useless features?

Feature selection can often result in better performing (less overfit), easier to understand, and faster model.

How do we carry out feature selection?

- There are a number of ways.
- You could use domain knowledge to discard features.
- We are briefly going to look at two automatic feature selection methods from `sklearn`:
 - Model-based selection
 - Recursive feature elimination
 - Forward selection
- Very related to looking at feature importances.

```
In [24]: 1 from sklearn.datasets import load_breast_cancer
2
3 cancer = load_breast_cancer()
4 X_train, X_test, y_train, y_test = train_test_split(
5     cancer.data, cancer.target, random_state=0, test_size=0.5
6 )
```

```
In [25]: 1 X_train.shape
```

```
Out[25]: (284, 30)
```

```
In [26]: 1 pipe_lr_all_feats = make_pipeline(StandardScaler(), LogisticRegression(
2 pipe_lr_all_feats.fit(X_train, y_train)
3 pd.DataFrame(
4     cross_validate(pipe_lr_all_feats, X_train, y_train, return_train_sc
5 ).mean()
```

```
Out[26]: fit_time      0.002101
score_time      0.000177
test_score      0.968233
train_score      0.987681
dtype: float64
```

Model-based selection

- Use a supervised machine learning model to judge the importance of each feature.

- Keep only the most important ones.
- Supervised machine learning model used for feature selection can be different than the one used as the final estimator.
- Use a model which has some way to calculate feature importances.

- To use model-based selection, we use `SelectFromModel` transformer.
- It selects features which have the feature importances greater than the provided threshold.
- Below I'm using `RandomForestClassifier` for feature selection with threshold "median" of feature importances.
- Approximately how many features will be selected?

In [27]:

```

1 from sklearn.ensemble import RandomForestClassifier
2 from sklearn.feature_selection import SelectFromModel
3
4 select_rf = SelectFromModel(
5     RandomForestClassifier(n_estimators=100, random_state=42), threshold
6 )

```

We can put the feature selection transformer in a pipeline.

In [28]:

```

1 pipe_lr_model_based = make_pipeline(
2     StandardScaler(), select_rf, LogisticRegression(max_iter=1000)
3 )
4
5 pd.DataFrame(
6     cross_validate(pipe_lr_model_based, X_train, y_train, return_train_
7 ).mean()

```

Out[28]:

fit_time	0.072111
score_time	0.005723
test_score	0.950564
train_score	0.974480
dtype:	float64

In [29]:

```

1 pipe_lr_model_based.fit(X_train, y_train)
2 pipe_lr_model_based.named_steps["selectfrommodel"].transform(X_train).shape

```

Out[29]:

(284, 15)

Similar results with only 15 features instead of 30 features.

Recursive feature elimination (RFE)

- Build a series of models
- At each iteration, discard the least important feature according to the model.
- Computationally expensive
- Basic idea
 - fit model
 - find least important feature

- remove
- iterate.

RFE algorithm

1. Decide k , the number of features to select.
2. Assign importances to features, e.g. by fitting a model and looking at `coef_` or `feature_importances_`.
3. Remove the least important feature.
4. Repeat steps 2-3 until only k features are remaining.

Note that this is **not** the same as just removing all the less important features in one shot!

In [30]:

```
1 scaler = StandardScaler()
2 X_train_scaled = scaler.fit_transform(X_train)
```

In [31]:

```
1 from sklearn.feature_selection import RFE
2
3 # create ranking of features
4 rfe = RFE(LogisticRegression(), n_features_to_select=5)
5 rfe.fit(X_train_scaled, y_train)
6
7 print(rfe.ranking_)
8 pd.DataFrame({
9     'feature': list(cancer.feature_names),
10    'rank': rfe.ranking_
11 })
```

```
[16 12 19 13 23 20 10 1 9 22 2 25 5 7 15 4 26 18 21 8 1 1 1 6
 14 24 3 1 17 11]
```

Out[31]:

	feature	rank
0	mean radius	16
1	mean texture	12
2	mean perimeter	19
3	mean area	13
4	mean smoothness	23
5	mean compactness	20
6	mean concavity	10
7	mean concave points	1
8	mean symmetry	9
9	mean fractal dimension	22
10	radius error	2
11	texture error	25
12	perimeter error	5
13	area error	7
14	smoothness error	15
15	compactness error	4
16	concavity error	26
17	concave points error	18
18	symmetry error	21
19	fractal dimension error	8
20	worst radius	1
21	worst texture	1
22	worst perimeter	1
23	worst area	6
24	worst smoothness	14
25	worst compactness	24
26	worst concavity	3
27	worst concave points	1
28	worst symmetry	17
29	worst fractal dimension	11

Question: what was the first feature to be eliminated? And the last?

In []:

1

In [32]:

1 print(rfe.support_)

```
[False False False False False False True False False False False
 False False False False False False False True True True False
 False False False True False False]
```

In [33]:

1 print("selected features: ", cancer.feature_names[rfe.support_])

```
selected features:  ['mean concave points' 'worst radius' 'worst texture'
 'worst perimeter'
 'worst concave points']
```

- How do we know what value to pass to `n_features_to_select` ?

- Use `RFECV` which uses cross-validation to select number of features.

In [34]:

```
1 from sklearn.feature_selection import RFECV
2
3 rfe_cv = RFECV(LogisticRegression(max_iter=2000), cv=10)
4 rfe_cv.fit(X_train_scaled, y_train)
5 print(rfe_cv.support_)
6 print(cancer.feature_names[rfe_cv.support_])
```

```
[False True False True False False True True True False True False
 True True False True False False False True True True True True
 False False True True False True]
['mean texture' 'mean area' 'mean concavity' 'mean concave points'
 'mean symmetry' 'radius error' 'perimeter error' 'area error'
 'compactness error' 'fractal dimension error' 'worst radius'
 'worst texture' 'worst perimeter' 'worst area' 'worst concavity'
 'worst concave points' 'worst fractal dimension']
```

In [35]:

```
1 rfe_pipe = make_pipeline(
2     StandardScaler(),
3     RFECV(LogisticRegression(max_iter=2000), cv=10),
4     RandomForestClassifier(n_estimators=100, random_state=42),
5 )
6
7 pd.DataFrame(cross_validate(rfe_pipe, X_train, y_train, return_train_sc
```

```
Out[35]: fit_time      0.491919
score_time      0.003431
test_score      0.943609
train_score      1.000000
dtype: float64
```

- Slow because there is cross validation within cross validation
- No improvement in scores (it decreases a bit) compared to all features on this toy case

Search and score

- Define a **scoring function** $f(S)$ that measures the quality of the set of features S .
- Now **search** for the set of features S with the best score.

General idea of search and score methods

- Example: Suppose you have three features: A, B, C
 - Compute **score** for $S = \{\}$
 - Compute **score** for $S = \{A\}$
 - Compute **score** for $S = \{B\}$
 - Compute **score** for $S = \{C\}$
 - Compute **score** for $S = \{A, B\}$
 - Compute **score** for $S = \{A, C\}$
 - Compute **score** for $S = \{B, C\}$
 - Compute **score** for $S = \{A, B, C\}$
- Return S with the best score.
- How many distinct combinations we have to try out?

Forward or backward selection

- Also called wrapper methods
- Shrink or grow feature set by removing or adding one feature at a time
- Makes the decision based on whether adding/removing the feature improves the CV score or not

iteration	current round scores	candidates	selected features	best score(error)
1.	$\text{score}(x_1) = 0.40$ $\text{Score}(x_2) = 0.39$ $\text{Score}(x_3) = 0.43$ $\checkmark \text{Score}(x_4) = 0.30$	$\{x_1, x_2, x_3, x_4\}$	$\{\}$	∞
2.	$\text{score}(x_1, x_4) = 0.35$ $\checkmark \text{Score}(x_2, x_4) = 0.28$ $\text{Score}(x_3, x_4) = 0.4$	$\{x_1, x_2, x_3\}$	$\{x_4\}$	0.30
3	$\text{score}(x_1, x_4, x_2) = 0.29$ $\text{Score}(x_3, x_4, x_2) = 0.30$	$\{x_1, x_3\}$	$\{x_4, x_2\}$	0.28

No score is less than the best score (error) so STOP.

```
In [36]: 1 from sklearn.feature_selection import SequentialFeatureSelector
2
3 pipe_forward = make_pipeline(
4     StandardScaler(),
5     SequentialFeatureSelector(LogisticRegression(max_iter=1000), direct=
6         RandomForestClassifier(n_estimators=100, random_state=42),
7     )
8 pd.DataFrame(
9     cross_validate(pipe_forward, X_train, y_train, return_train_score=True,
10 ).mean()
```

```
Out[36]: fit_time      2.154436
score_time      0.003376
test_score       0.933020
train_score      1.000000
dtype: float64
```

```
In [37]: 1 pipe_forward = make_pipeline(
2     StandardScaler(),
3     SequentialFeatureSelector(
4         LogisticRegression(max_iter=1000), direction="backward", n_feat=
5     ),
6     RandomForestClassifier(n_estimators=100, random_state=42),
7 )
8 pd.DataFrame(
9     cross_validate(pipe_forward, X_train, y_train, return_train_score=True,
10 ).mean()
```

```
Out[37]: fit_time      2.720003
score_time      0.003380
test_score       0.950627
train_score      1.000000
dtype: float64
```

Other ways to search

- Stochastic local search
 - Inject randomness so that we can explore new parts of the search space
 - Simulated annealing
 - Genetic algorithms

Warnings about feature selection

- A feature's relevance is only defined in the context of other features.
 - Adding/removing features can make features relevant/irrelevant.
- If features can be predicted from other features, you cannot know which one to pick.
- Relevance for features does not have a causal relationship.
- Don't be overconfident.
 - The methods we have seen probably do not discover the ground truth and how the world really works.
 - They simply tell you which features help in predicting y_i for the data you have.

(Optional) Problems with feature selection

- The term 'relevance' is not clearly defined.
- What things can go wrong with feature selection?
- Attribution: From CPSC 340.

Example: Is "Relevance" clearly defined?

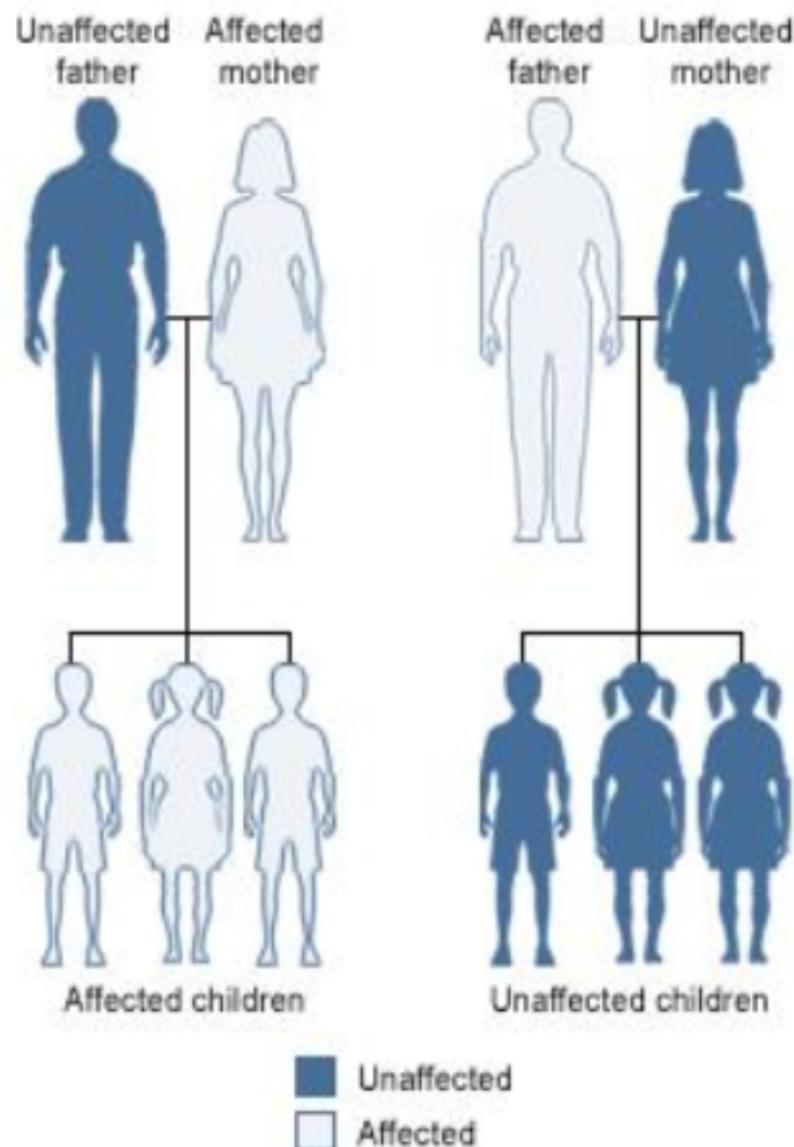
- Consider a supervised classification task of predicting whether someone has particular genetic variation (SNP)

sex	biological dad	biological mom	SNP
F	0	1	1
M	1	0	0
F	0	0	0
F	1	1	1

- True model: You almost have the same value as your biological mom.

Is "Relevance" clearly defined?

- True model: You almost have the same value for SNP as your biological mom.
 - ($\text{SNP} = \text{biological mom}$) with very high probability
 - ($\text{SNP} \neq \text{biological mom}$) with very low probability



Is "Relevance" clearly defined?

- What if "mom" feature is repeated?
- Should we pick both? Should we pick one of them because it predicts the other?
- Dependence, collinearity for linear models
 - If a feature can be predicted from the other, don't know which one to pick.

sex	biological dad	biological mom	mom2	SNP
-----	----------------	----------------	------	-----

Is "Relevance" clearly defined?

- What if we add (maternal) "grandma" feature?
- Is it relevant?
 - We can predict SNP accurately using this feature
- Conditional independence
 - But grandma is irrelevant given biological mom feature
 - Relevant features may become irrelevant given other features

sex	biological dad	biological mom	grandma	SNP
F	0	1	1	1
M	1	0	0	0
F	0	0	0	0
F	1	1	1	1

Is "Relevance" clearly defined?

- What if we do not know biological mom feature and we just have grandma feature
- It becomes relevant now.
 - Without mom feature this is the best we can do.
 - Features can become relevant due to missing information

sex	biological dad	grandma	SNP
F	0	1	1
M	1	0	0
F	0	0	0
F	1	1	1

Is "Relevance" clearly defined?

- Are there any relevant features now?
- They may have some common maternal ancestor.
- What if mom likes dad because they share SNP?
- General problem (Confounding)
 - Hidden features can make irrelevant features relevant.

sex	biological dad	SNP
F	0	1
M	1	0
F	0	0
F	1	1

Is "Relevance" clearly defined?

- Now what if we have "sibling" feature?
- The feature is relevant in predicting SNP but not the cause of SNP.
- General problem (non causality)
 - the relevant feature may not be causal

sex	biological dad	sibling	SNP
F	0	1	1
M	1	0	0
F	0	0	0
F	1	1	1

Is "Relevance" clearly defined?

- What if you are given "baby" feature?
- Now the sex feature becomes relevant.
 - "baby" feature is relevant when $\text{sex} == \text{F}$
- General problem (context specific relevance)
 - adding a feature can make an irrelevant feature relevant

sex	biological dad	baby	SNP
F	0	1	1
M	1	1	0
F	0	0	0
F	1	1	1

Warnings about feature selection

- A feature is only relevant in the context of other features.
 - Adding/removing features can make features relevant/irrelevant.
- Confounding factors can make irrelevant features the most relevant.
- If features can be predicted from other other features, you cannot know which one to pick.
- Relevance for features does not have a causal relationship.
- Is feature selection completely hopeless?
 - It is messy but we still need to do it. So we try to do our best!

General advice on finding relevant features

- Try forward selection.
- Try other feature selection methods (e.g., RFE, simulated annealing, genetic algorithms)
- Talk to domain experts; they probably have an idea why certain features are relevant.
- Don't be overconfident.
 - The methods we have seen probably do not discover the ground truth and how the world really works.
 - They simply tell you which features help in predicting y_i .

Relevant resources

- [Genome-wide association study](https://en.wikipedia.org/wiki/Genome-wide_association_study) (https://en.wikipedia.org/wiki/Genome-wide_association_study)
- [sklearn feature selection](https://scikit-learn.org/stable/modules/feature_selection.html) (https://scikit-learn.org/stable/modules/feature_selection.html)
- [PyData: A Practical Guide to Dimensionality Reduction Techniques](https://www.youtube.com/watch?v=ioXKxulmwVQ) (<https://www.youtube.com/watch?v=ioXKxulmwVQ>)

True/False questions for class discussion

1. Simple association-based feature selection approaches do not take into account the interaction between features.
2. You can carry out feature selection using linear models by pruning the features which have very small weights (i.e., coefficients less than a threshold).
3. Forward search is guaranteed to find the best feature set.
4. The order of features removed given by `rfe.ranking_` is the same as the order of original feature importances given by the model.

In []:

1

CPSC 330

Applied Machine Learning

Lecture 14: Clustering

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

```
In [133]: 1 import os
2 import random
3 import sys
4 import time
5
6 import numpy as np
7
8 sys.path.append("../code/.")
9 import matplotlib.pyplot as plt
10 import mglearn
11 import seaborn as sns
12 from plotting_functions import *
13 from plotting_functions_unsup import *
14 from sklearn import cluster, datasets, metrics
15 from sklearn.compose import ColumnTransformer, make_column_transformer
16 from sklearn.datasets import make_blobs
17 from sklearn.decomposition import PCA
18 from sklearn.pipeline import Pipeline, make_pipeline
19 from sklearn.preprocessing import StandardScaler
20 from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer
21 import pandas as pd
22
23 plt.style.use("seaborn-v0_8")
24
25 %matplotlib inline
```

Learning outcomes

From this lecture, students are expected to be able to:

- Explain the motivation and potential applications of clustering.
- Define the clustering problem.
- Explain the K-Means algorithm.

- Apply `sklearn`'s KMeans algorithm.
- Apply the Elbow method and Silhouette method to choose the number of clusters.
- Use clustering for customer segmentation problem.
- Interpret the clusters discovered by K-Means.

Unsupervised learning

Types of machine learning from

Recall the typical learning problems we discussed at the beginning of the course.

- Supervised learning ([Gmail spam filtering \(<https://support.google.com/a/answer/2368132?hl=en>\)](https://support.google.com/a/answer/2368132?hl=en)
 - Training a model from input data and its corresponding targets to predict targets for new examples. (571, 572, 573)
- **Unsupervised learning** (this course) ([Google News \(<https://news.google.com/>\)](https://news.google.com/)
 - Training a model to find patterns in a dataset, typically an unlabeled dataset.
- Reinforcement learning ([AlphaGo \(<https://deepmind.com/research/case-studies/alphago-the-story-so-far>\)](https://deepmind.com/research/case-studies/alphago-the-story-so-far)
 - A family of algorithms for finding suitable actions to take in a given situation in order to maximize a reward.
- **Recommendation systems** ([Amazon item recommendation system \(<https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf>\)](https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf)
 - Predict the "rating" or "preference" a user would give to an item.

Supervised learning

- Training data comprises a set of observations (X) and their corresponding targets (y).
- We wish to find a model function f that relates X to y .
- Then use that model function to predict the targets of new examples.
- We have been working with this set up so far.

Training data

X	y
-----	-----

Unseen test data

X	y
-----	-----

Labeled vs. Unlabeled data

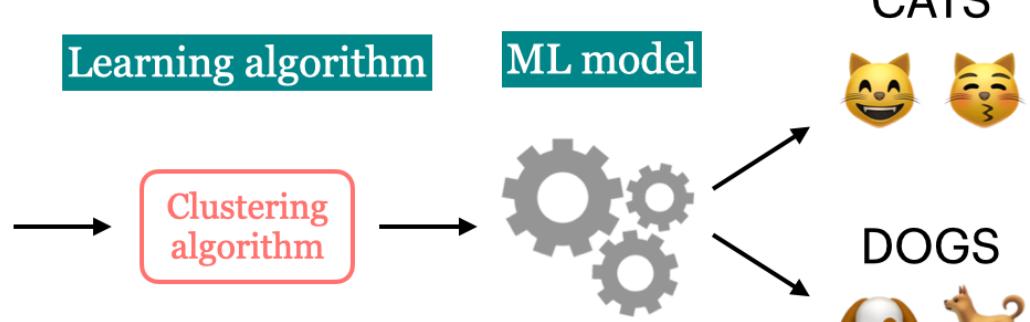
- If you have access to labeled training data, you're in the "supervised" setting.
- You know what to do in that case.
- Unfortunately, getting large amount of labeled training data is often difficult, expensive, or simply impossible in some cases.
- Can you still make sense of the data even though you do not have the labels?
- Yes! At least to a certain extent!

Unsupervised learning

- Training data consists of observations (X) without any corresponding targets.
- Unsupervised learning could be used to group similar things together in X .

Training data

X
🐱
😺
...
🐶
🐕

Learning algorithm**Example: Supervised vs unsupervised learning**

- In supervised learning, we are given features X and target y .

Dataset 1		Dataset2		
x_1	y	x_1	x_2	y
101.0	Sick	-2.68	0.32	class 1
98.5	Not Sick	-2.71	-0.18	class 1
93.8	Sick	1.28	0.69	class 2
104.3	Sick	0.93	0.32	class 2
98.6	Not Sick	1.39	-0.28	class 3

- In unsupervised learning, we are only given features X .

Dataset 1		Dataset 2	
x_1	x_2	x_1	x_2
101.0	0.32	-2.68	0.32
98.5	-0.18	-2.71	-0.18
93.8	0.69	1.28	0.69
104.3	0.32	0.93	0.32
98.6	-0.28	1.39	-0.28

An example with sklearn toy dataset

In [134]:

```

1 ## Iris dataset
2 iris = datasets.load_iris() # loading the iris dataset
3 features = iris.data[:, 2:4] # only consider two features for visualization
4 labels = iris.target_names[
5     iris.target
6 ] # get the targets, in this case the types of the Iris flower
7
8 iris_df = pd.DataFrame(features, columns=iris.feature_names[2:])
9 iris_df.head()

```

Out[134]:

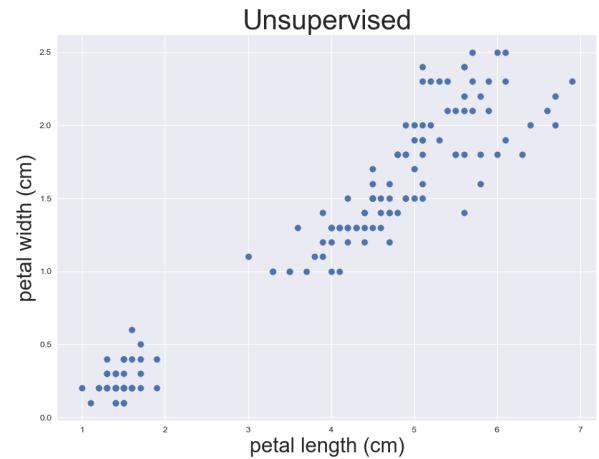
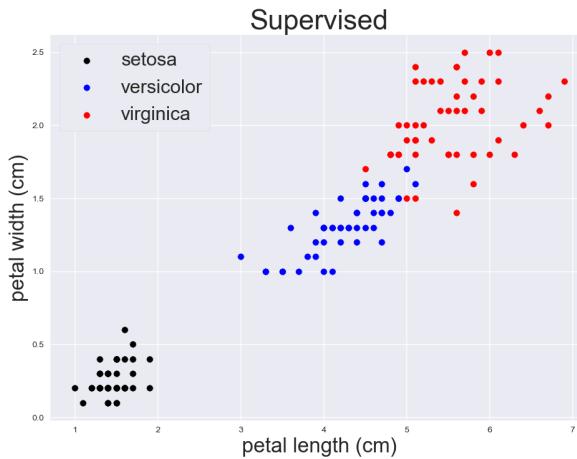
	petal length (cm)	petal width (cm)
0	1.4	0.2
1	1.4	0.2
2	1.3	0.2
3	1.5	0.2
4	1.4	0.2

	petal length (cm)	petal width (cm)
0	1.4	0.2
1	1.4	0.2
2	1.3	0.2
3	1.5	0.2
4	1.4	0.2

```
In [135]: 1 np.unique(labels)
```

```
Out[135]: array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
In [136]: 1 iris_df["target"] = labels
2 plot_sup_x_unsup(iris_df, 8, 8)
```



- In case of supervised learning, we're given X and y (showed with different colours in the plot above).
- In case of unsupervised learning, we're only given X and the goal is to identify the underlying structure in data.

Can we learn without targets?

- Yes, but the learning will be focused on finding the underlying structures of the inputs themselves (rather than finding the function f between input and output like we did in supervised learning models).
- Examples:
 - Clustering
 - Dimensionality Reduction (we won't cover it in this course)

Clustering motivation

Why clustering?

- Most of the data out there is unlabeled.
- Getting labeled training data is often difficult, expensive, or simply impossible in some cases.
- Can we extract some useful information from unlabeled data?

- The most intuitive way is to group similar examples together to get some insight into the data even though we do not have the targets

Clustering

Clustering is the task of partitioning the dataset into groups called clusters.

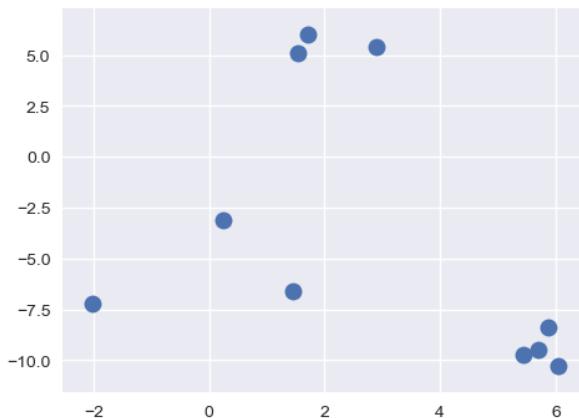
The goal of clustering is to discover underlying groups in a given dataset such that:

- examples in the same group are as similar as possible;
- examples in different groups are as different as possible.

Input and possible output

In [137]:

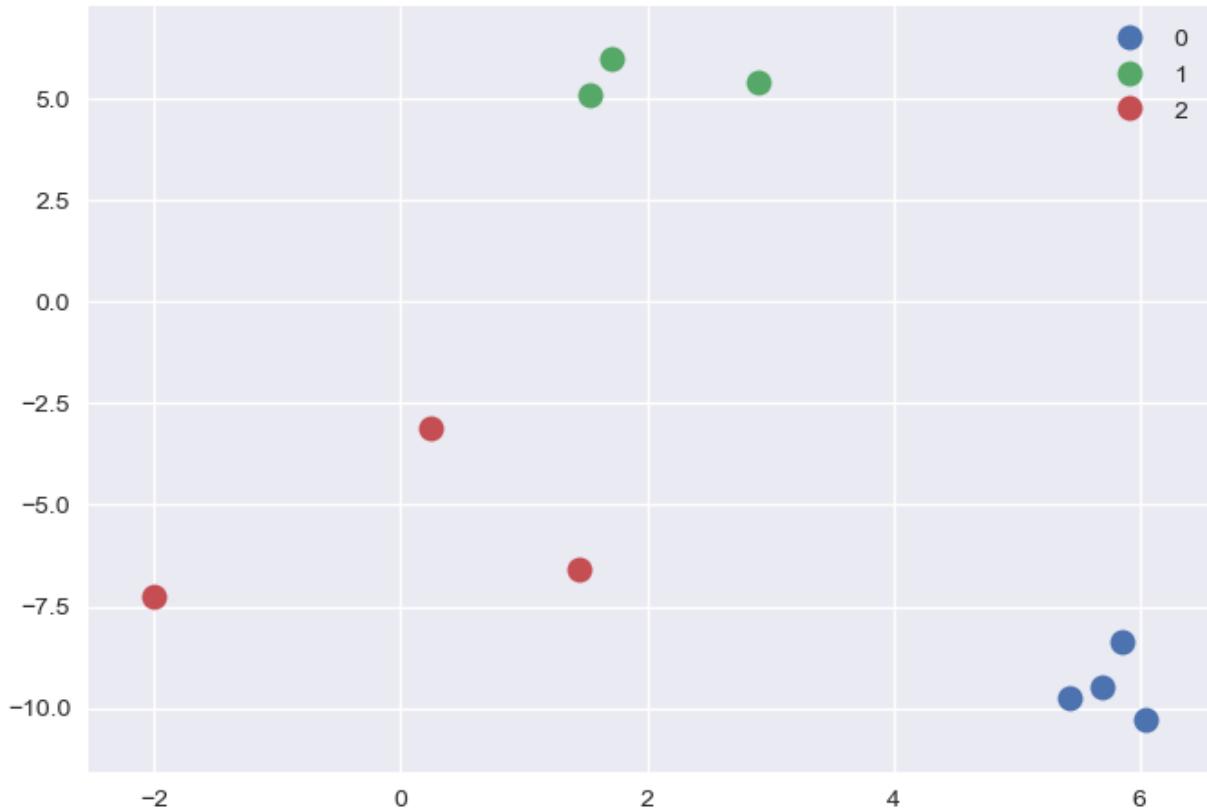
```
1 X, y = make_blobs(n_samples=10, centers=3, n_features=2, random_state=1)
2 fig, axes = plt.subplots(1, 2, figsize=(12, 4))
3 mglearn.discrete_scatter(X[:, 0], X[:, 1], markers="o", ax=axes[0])
4 mglearn.discrete_scatter(X[:, 0], X[:, 1], y, markers="o", ax=axes[1]);
```



Think of clustering as colouring the points (e.g., blue, red, green) such that points with the same color are close to each other.

In [138]:

```
1 mglearn.discrete_scatter(X[:, 0], X[:, 1], y, markers="o")
2 plt.legend();
```



Is there a notion of "correct" grouping?

- Very often we do not know how many clusters are there in the data or if there are any clusters at all. In real-world data, clusters are rarely as clear as in our toy example above.
- There is a notion of coherent and optimal (in some sense) clusters but there is no absolute truth here.

Example 1

Which of the following grouping of emoticons is the "correct" grouping?

Categorization 1

Both seem reasonable!

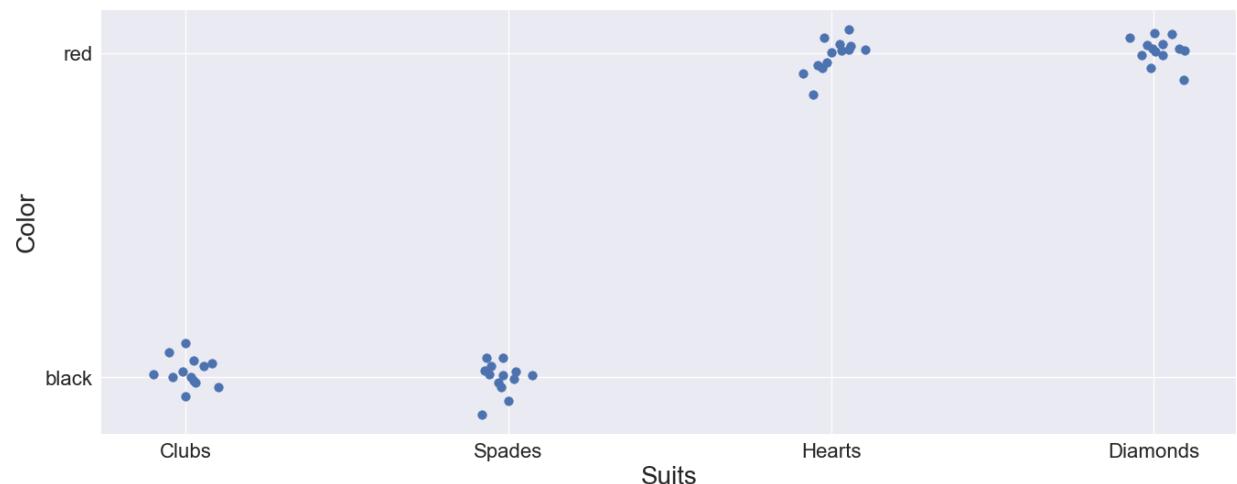
Example 2

How would you group a deck of cards?

- In a deck of cards:
 1. We have two colors: black and red;
 2. We have four suits: Clubs, Spades, Hearts, and Diamonds;
 3. We have 13 values: { A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K }.
- What are the "true" clusters here?

Should we cluster by suits: Clubs, Spades, Hearts, and Diamonds?

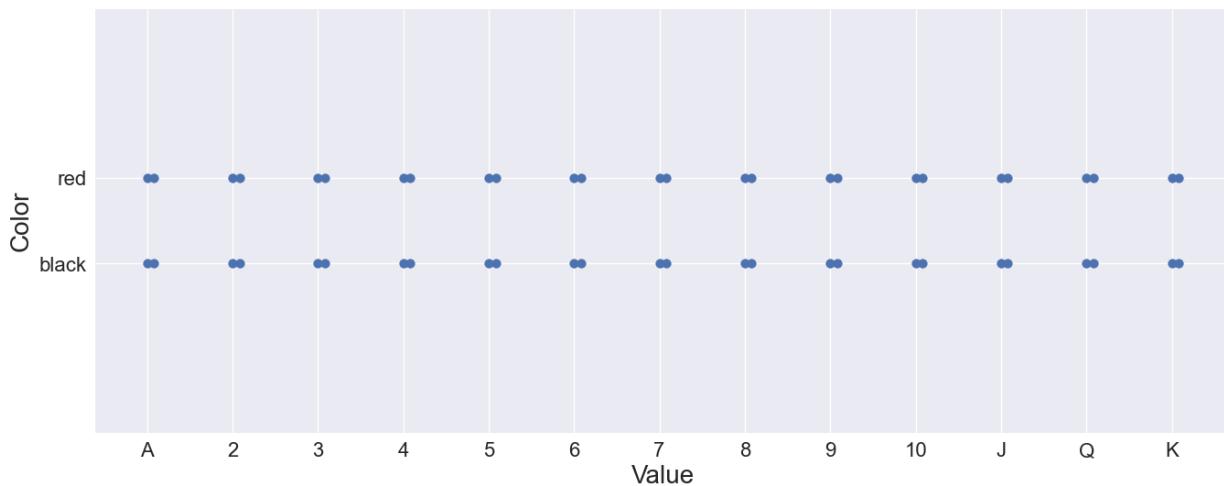
In [139]: 1 plot_deck(group_by="suits", w=16, h=6)



Should we cluster by colors?

In [140]:

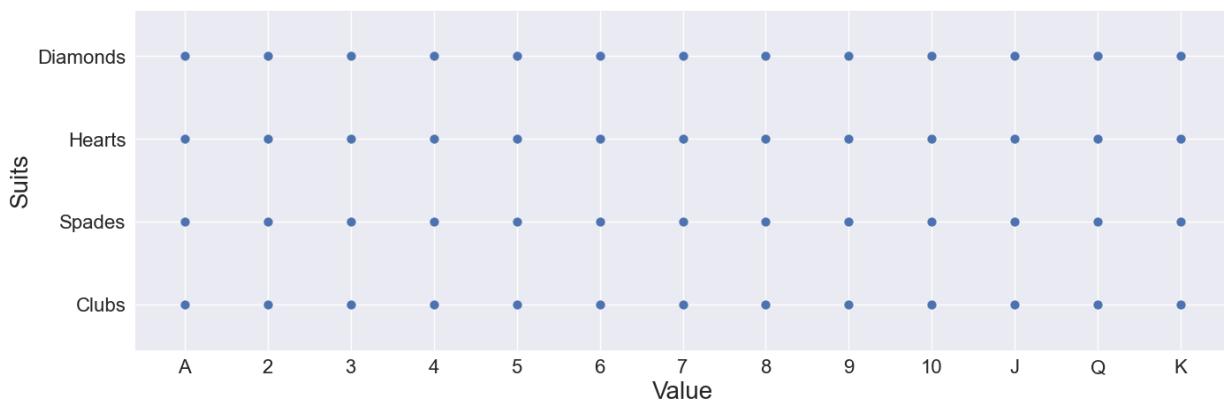
```
1 plot_deck(group_by="color", w=16, h=6)
```



Should we cluster by value?

In [141]:

```
1 plot_deck(group_by="cards", w=16, h=5)
```



- All these options seem reasonable.

Meaningful groups in clustering

- In clustering, meaningful groups are dependent on the **application**.
- It usually helps if we have some prior knowledge about the data and the problem.
- This makes it hard for us to objectively measure the quality of a clustering algorithm (or think about "true" clusters).

Common applications: Data exploration

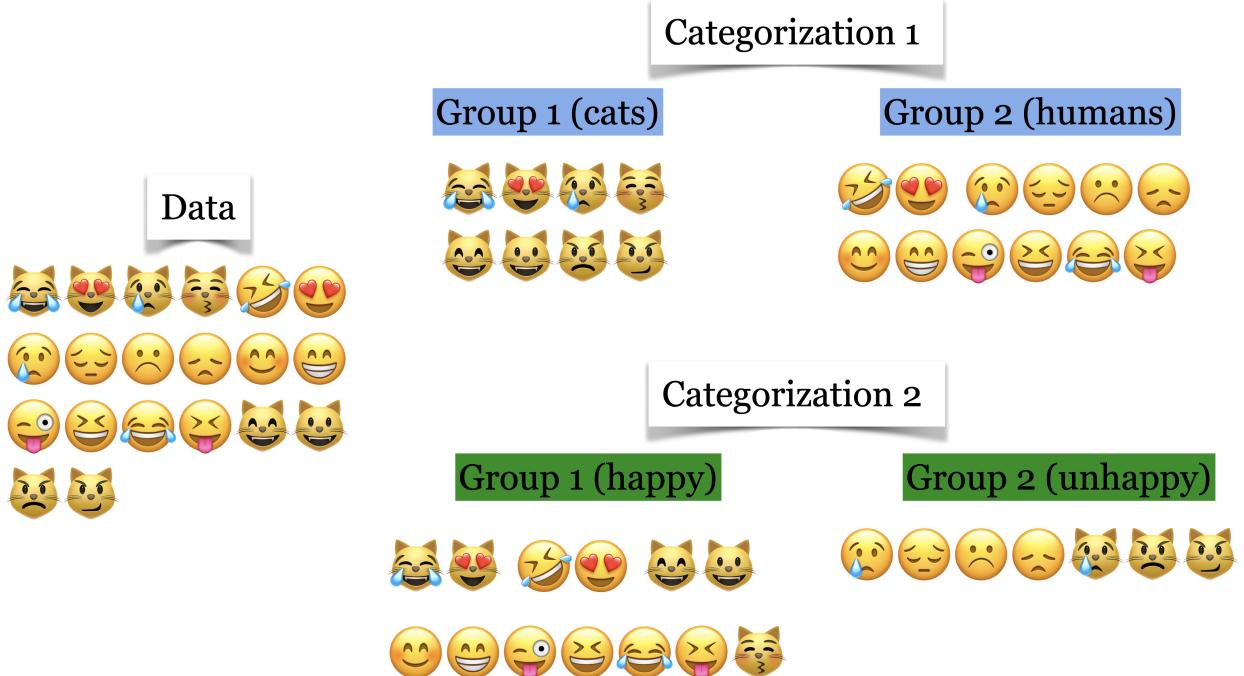
Although there is no notion of the "right" answer, we might still get something useful out of clustering. There are a number of common applications for clustering.

- Summarize or compress data.

- Partition the data into groups before further processing.
 - For instance, you could use it in supervised learning setting as follows. Carry out clustering and examine performance of your model on individual clusters. If the performance is lower on a particular cluster, you could either try building a separate model for that cluster and improve the overall performance of your supervised model.

Common applications: Customer segmentation

- Understand landscape of the market in businesses and craft targeted business or marketing strategies tailored for each group.



source (<https://www.youtube.com/watch?v=zPJtDohab-g&t=134s>)

Document clustering

Grouping articles on different topics from different news sources. For example, [Google News](https://news.google.com) (<https://news.google.com>).

Similarity and distances

- Clustering is based on the notion of similarity or distances between points.
 - How do we determine similarity between points in a multi-dimensional space?
 - Can we use something like k -NN for similarity?
 - Yes! That's a good start!
 - With k -NN we used Euclidean distances to find nearby points.
 - We can use the same idea for clustering!

K-Means clustering algorithm

K-Means clustering

One of the most commonly used clustering algorithm.

Input

- `x` → a set of data points
- `k` (or `k` or `n_clusters`) → number of clusters

Output

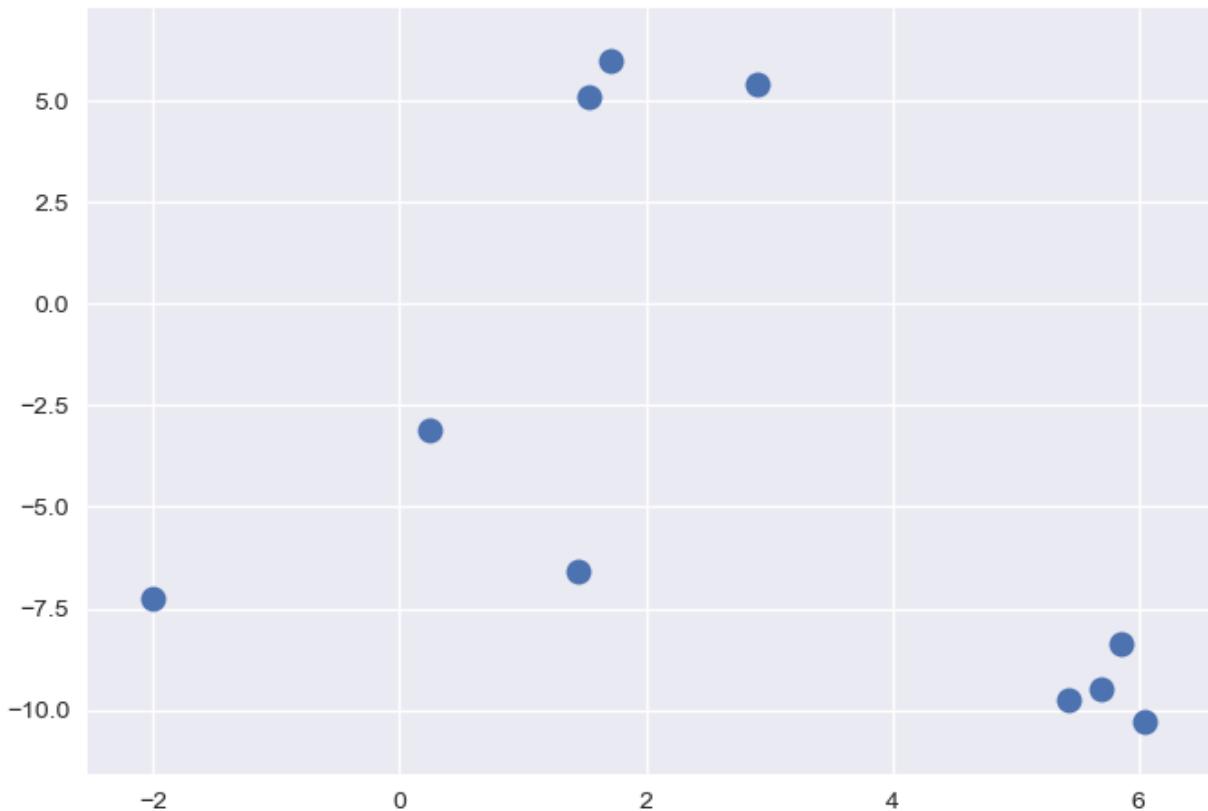
- `k` clusters (groups) of the data points

K-Means using `sklearn`

- Before understanding the algorithm, let's try it with `sklearn`.
- Consider the toy dataset above.
- For this toy dataset, the three clusters are pretty clear.

In [142]:

```
1 X, y = make_blobs(n_samples=10, centers=3, n_features=2, random_state=1)
2 mglearn.discrete_scatter(X[:, 0], X[:, 1], markers="o");
```



In [143]:

```
1 X
```

```
Out[143]: array([[ 5.69192445, -9.47641249],
       [ 1.70789903,  6.00435173],
       [ 0.23621041, -3.11909976],
       [ 2.90159483,  5.42121526],
       [ 5.85943906, -8.38192364],
       [ 6.04774884, -10.30504657],
       [-2.00758803, -7.24743939],
       [ 1.45467725, -6.58387198],
       [ 1.53636249,  5.11121453],
       [ 5.4307043 , -9.75956122]])
```

```
In [144]: 1 toy_df = pd.DataFrame(data=X, columns=["feat1", "feat2"])
2 toy_df
```

Out[144]:

	feat1	feat2
0	5.691924	-9.476412
1	1.707899	6.004352
2	0.236210	-3.119100
3	2.901595	5.421215
4	5.859439	-8.381924
5	6.047749	-10.305047
6	-2.007588	-7.247439
7	1.454677	-6.583872
8	1.536362	5.111215
9	5.430704	-9.759561

KMeans fit

Let's try `sklearn`'s `KMeans` algorithm on this dataset.

- We need to decide how many clusters we want. Here we are passing 3.
- We are only passing `X` because this is unsupervised learning; we do not have labels.

```
In [145]: 1 from sklearn.cluster import KMeans
2
3 kmeans = KMeans(n_clusters=3)
4 kmeans.fit(X)
5 # We are only passing X because this is unsupervised learning
```

Out[145]: `KMeans(n_clusters=3)`

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with [nbviewer.org](#).

predict of KMeans

- The output of `KMeans` is K clusters (groups) of the data points.
- Calling `predict` will give us the cluster assignment for each data point.

```
In [146]: 1 kmeans.predict(X)
```

Out[146]: `array([0, 1, 2, 1, 0, 0, 2, 2, 1, 0], dtype=int32)`

```
In [147]: 1 toy_df_cl = toy_df.copy()
2 toy_df_cl["cluster"] = kmeans.predict(toy_df.to_numpy())
3 toy_df_cl
```

```
Out[147]:   feat1      feat2  cluster
0    5.691924 -9.476412       0
1    1.707899  6.004352       1
2    0.236210 -3.119100       2
3    2.901595  5.421215       1
4    5.859439 -8.381924       0
5    6.047749 -10.305047      0
6   -2.007588 -7.247439       2
7    1.454677 -6.583872       2
8    1.536362  5.111215       1
9    5.430704 -9.759561       0
```

Cluster centers in K-Means

- In K-Means each cluster is represented by its cluster center.

```
In [148]: 1 kmeans.cluster_centers_
```

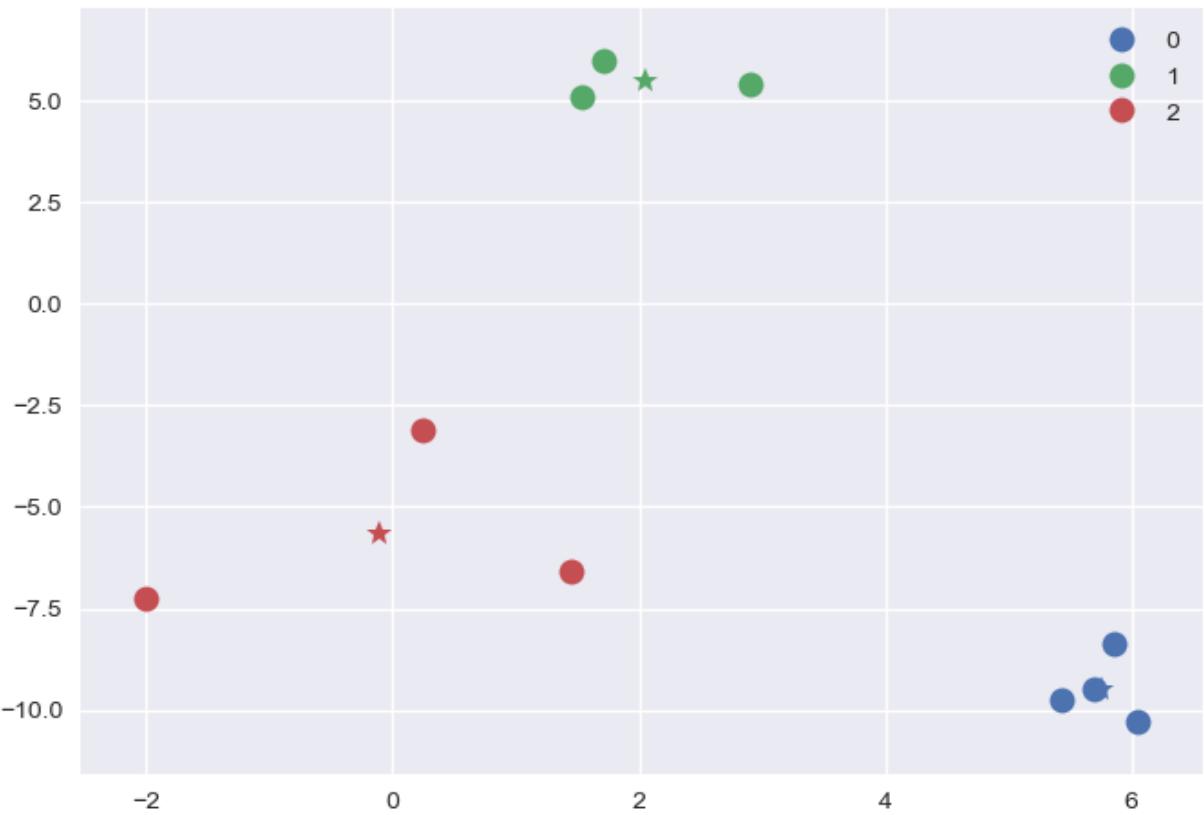
```
Out[148]: array([[ 5.75745416, -9.48073598],
                  [ 2.04861878,  5.51226051],
                  [-0.10556679, -5.65013704]])
```

In [149]:

```

1 mglearn.discrete_scatter(X[:, 0], X[:, 1], kmeans.labels_, markers="o")
2 plt.legend()
3 mglearn.discrete_scatter(
4     kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], [0, 1,
5 ]);

```



K-Means predictions on new examples

- We can also use `predict` on unseen examples!

In [150]:

```

1 new_examples = np.array([[-1, -5], [2, 5.0]])
2 kmeans.predict(new_examples)

```

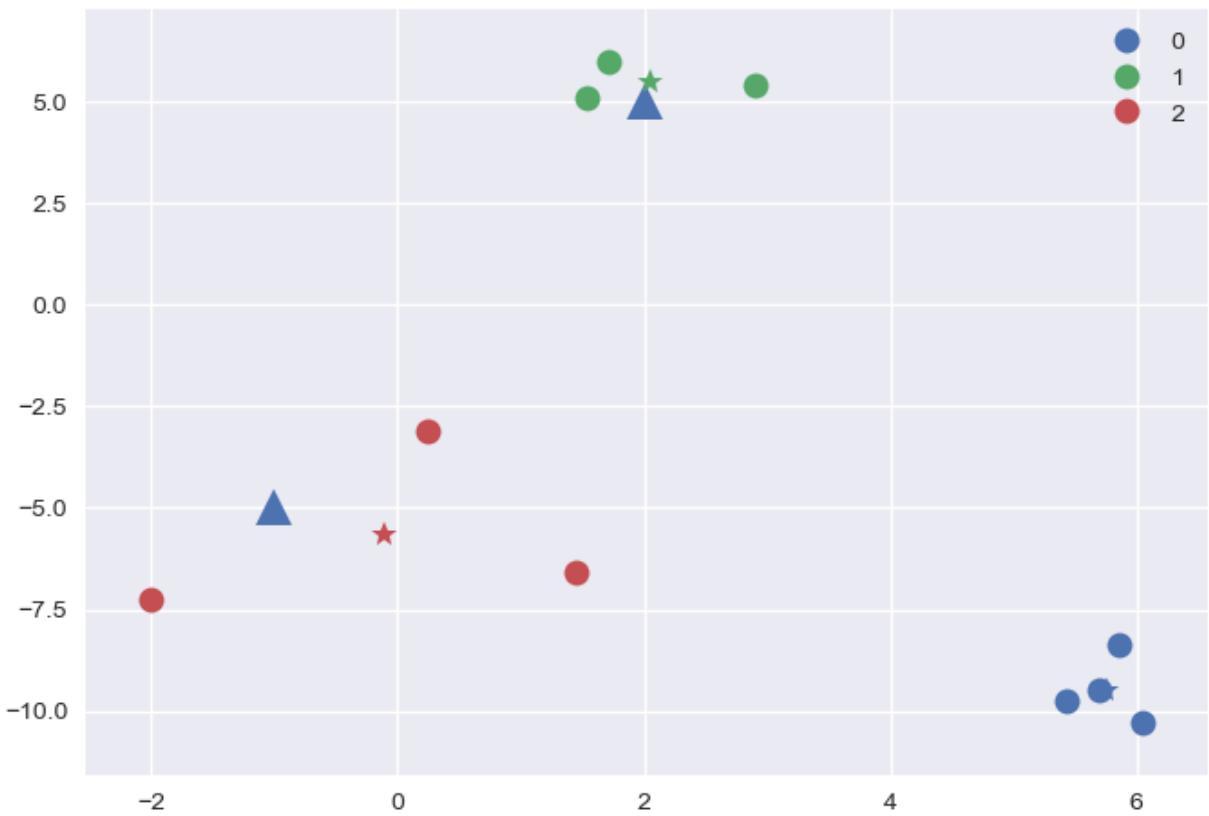
Out[150]: array([2, 1], dtype=int32)

In [151]:

```

1 mglearn.discrete_scatter(X[:, 0], X[:, 1], kmeans.labels_, markers="o")
2 plt.legend()
3 mglearn.discrete_scatter(new_examples[:, 0], new_examples[:, 1], marker
4 mglearn.discrete_scatter(
5     kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], [0, 1
6 );

```



K-Means algorithm: Main idea

- Represent each cluster by its cluster center and assign a cluster membership to each data point.

Chicken-and-egg problem!

- If we knew cluster centers, we can simply assign each point to its nearest center.
- Similarly, if we knew assignments, we can calculate cluster centers.
- But we do not know either 😕 .

A usual computer science answer to such problems is iterations!!

K-Means clustering algorithm

Input: Data points X and the number of clusters K

Initialization: K initial centers for the clusters

Iterative process:

repeat

- Assign each example to the closest center.
- Estimate new centers as *average* of observations in a cluster.

until **centers stop changing or maximum iterations have reached.**

Let's execute K-Means algorithm on our toy example.

Input

- The data points \mathbf{x}

```
In [152]: 1 n_examples = toy_df.shape[0]
2 print("Number of examples: ", n_examples)
3 X
```

Number of examples: 10

```
Out[152]: array([[ 5.69192445, -9.47641249],
       [ 1.70789903,  6.00435173],
       [ 0.23621041, -3.11909976],
       [ 2.90159483,  5.42121526],
       [ 5.85943906, -8.38192364],
       [ 6.04774884, -10.30504657],
       [-2.00758803, -7.24743939],
       [ 1.45467725, -6.58387198],
       [ 1.53636249,  5.11121453],
       [ 5.4307043 , -9.75956122]])
```

- Let K (number of clusters) be 3.

```
In [153]: 1 k = 3
```

Initialization

- Random initialization for K initial centers of the clusters.

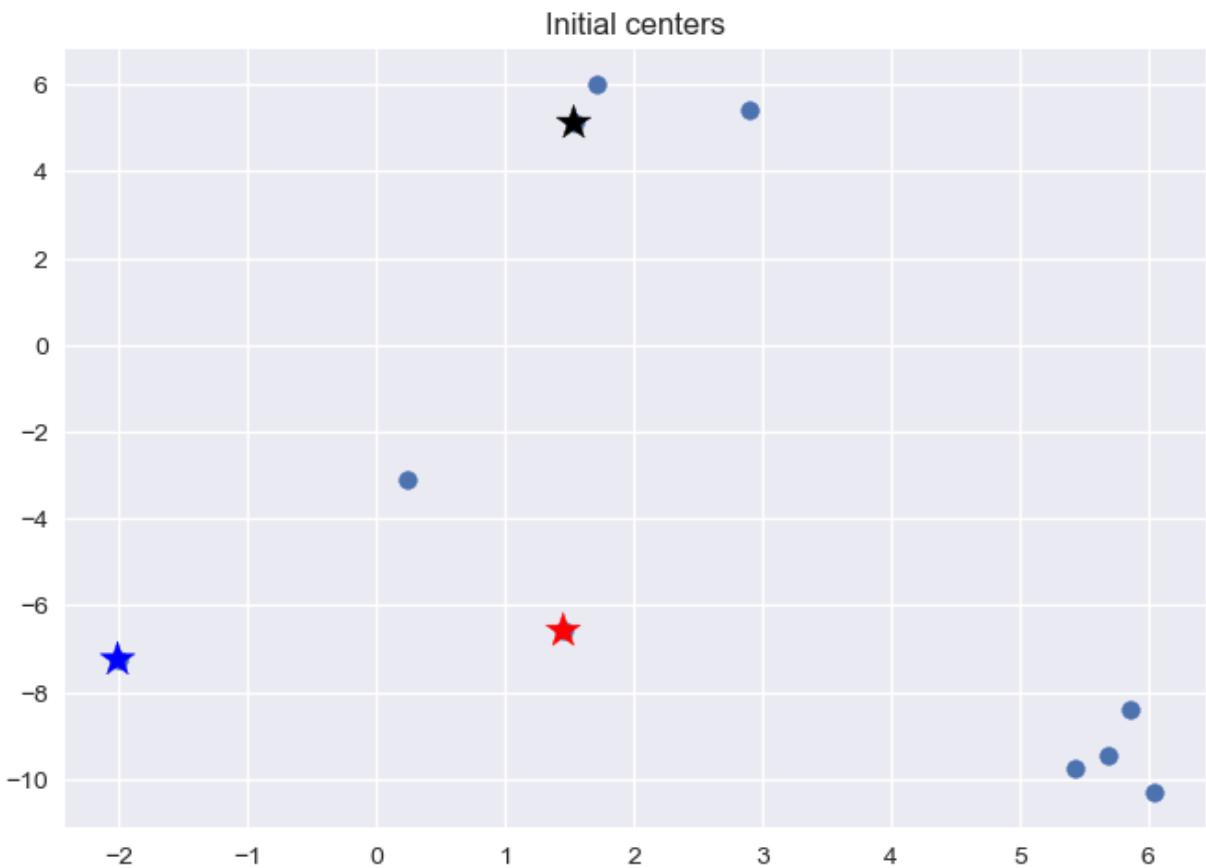
```
In [154]: 1 np.random.seed(seed=14)
2 centers_idx = np.random.choice(range(0, n_examples), size=k)
3 centers_df = toy_df.iloc[centers_idx]
4 centers = X[centers_idx]
5 colours = ["black", "blue", "red"]
```

In [155]:

```

1 plt.scatter(X[:, 0], X[:, 1], marker="o")
2 plt.scatter(centers[:, 0], centers[:, 1], c=colours, marker="*", s=200)
3 plt.title("Initial centers");

```



Iterative process

repeat

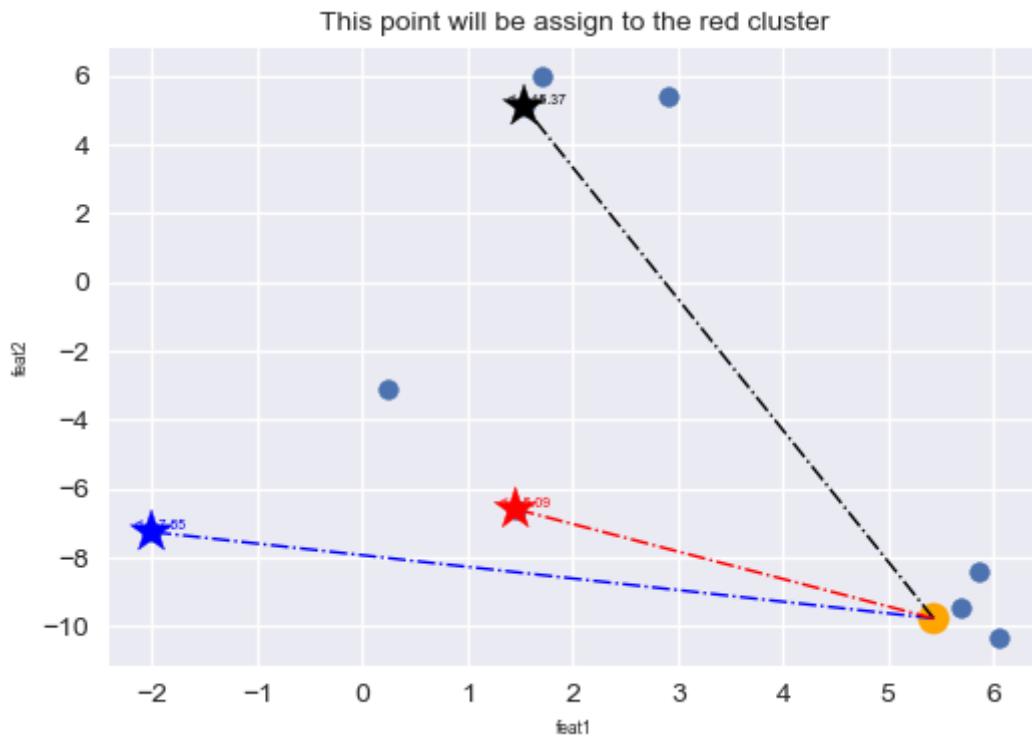
- Assign each example to the closest center. (`update_z`)
- Estimate new centers as average of observations in a cluster. (`update_centers`)

until **centers stop changing or maximum iterations have reached**.

How to find closest centers?

- First step in the iterative process is assigning examples to the closest center.
- Let's consider distance of an example to all centers and assign that example to the closest center.

```
In [156]: 1 plot_example_dist(toy_df, centers_df, 6, 4)
```



How to find closest centers?

- Similarly, we can make cluster assignments for all points by calculating distances of all examples to the centers and assigning it to the cluster with smallest distance.

```
In [157]: 1 from sklearn.metrics import euclidean_distances
```

```
2
3
4 def update_Z(X, centers):
5     """
6         returns distances and updated cluster assignments
7     """
8     dist = euclidean_distances(X, centers)
9     return dist, np.argmin(dist, axis=1)
```

How to update centers?

- With the new cluster assignments for our data points, we update cluster centers.
- New cluster centers are means of data points in each cluster.

In [158]:

```

1 def update_centers(X, z, old_centers, k):
2     """
3     returns new centers
4     """
5     new_centers = old_centers.copy()
6     for kk in range(k):
7         new_centers[kk] = np.mean(X[z == kk], axis=0)
8     return new_centers

```

Iteration 1: Step 1

- Assign each example to the closest cluster center.

In [159]:

```

1 dist, z = update_z(X, centers)
2 z

```

Out[159]: array([2, 0, 2, 0, 2, 2, 1, 2, 0, 2])

- This is the current cluster assignment.

In [160]:

```
1 plot_current_assinment(X, z, centers)
```



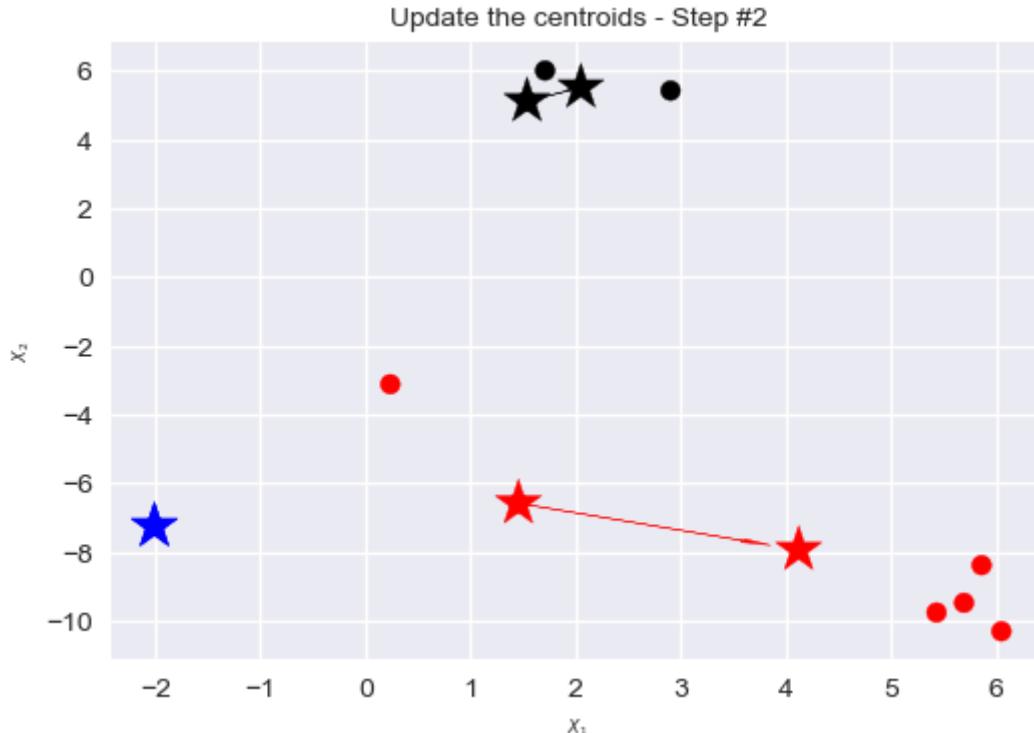
Iteration 1: Step 2

- Estimate new centers as *average* of observations in a cluster.

```
In [161]: 1 new_centers_it1 = update_centers(X, z, centers, k)
```

- This is how the centers moved in this iteration.

```
In [162]: 1 plot_update_centroid(toy_df, 6, 4, new_centers_it1, centers, dist)
```



Iteration 2: step 1

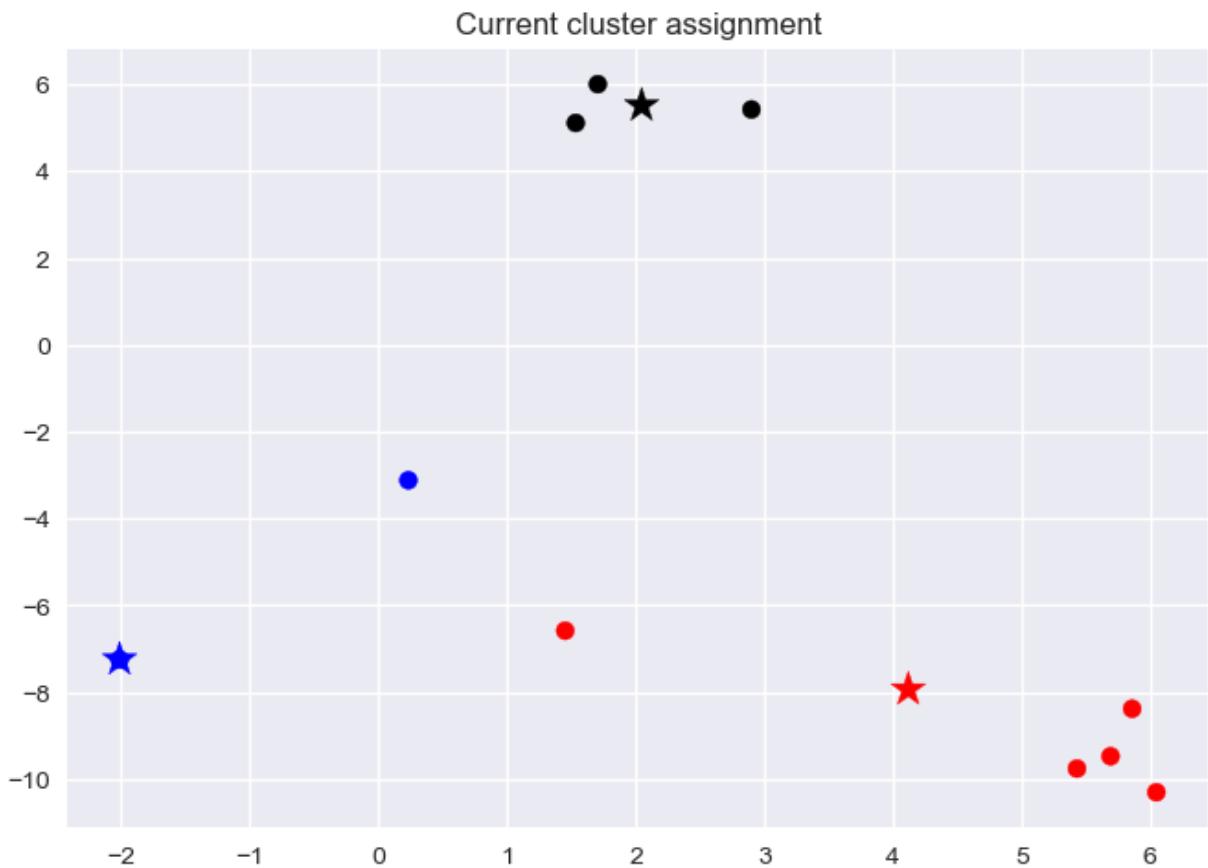
- Assign each example to the closest cluster center.

```
In [163]: 1 dist, z = update_z(X, new_centers_it1)
2 z
```

```
Out[163]: array([2, 0, 1, 0, 2, 2, 1, 2, 0, 2])
```

- This is the current cluster assignment.

```
In [164]: 1 plot_current_assinment(X, z, new_centers_it1)
```



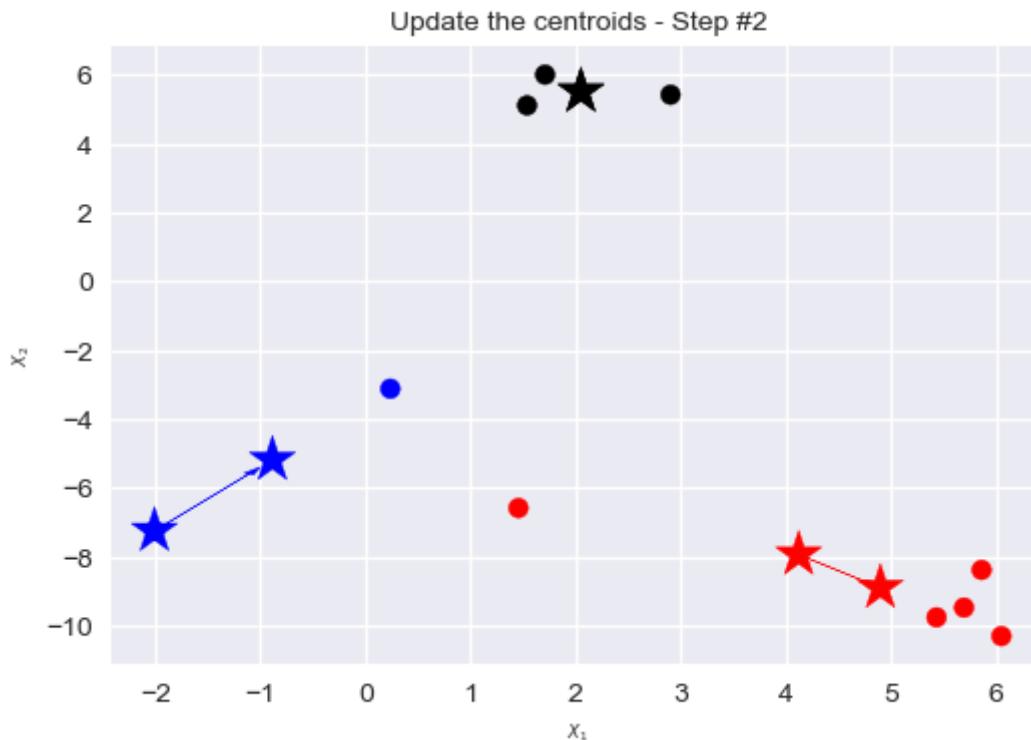
Iteration 2: step 2

- Estimate new centers as *average* of observations in a cluster.

```
In [165]: 1 new_centers_it2 = update_centers(X, z, new_centers_it1, k)
```

- This is how the centers moved in this iteration.

```
In [166]: 1 plot_update_centroid(toy_df, 6, 4, new_centers_it2, new_centers_it1, di
```



Iteration 3: step 1

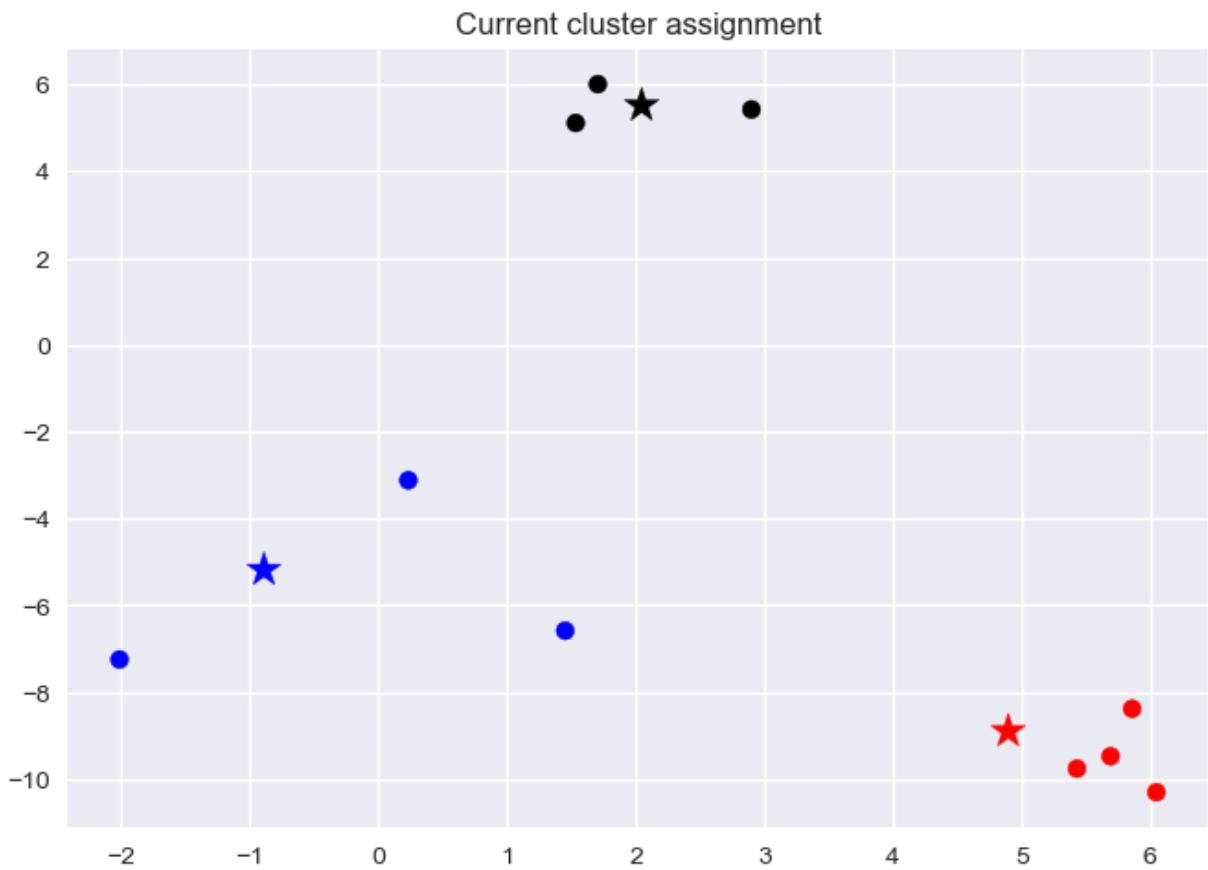
- Assign each example to the closest cluster center.

```
In [167]: 1 dist, z = update_z(X, new_centers_it2)
2 z
```

```
Out[167]: array([2, 0, 1, 0, 2, 2, 1, 1, 0, 2])
```

- This is the current cluster assignment.

```
In [168]: 1 plot_current_assinment(X, z, new_centers_it2)
```



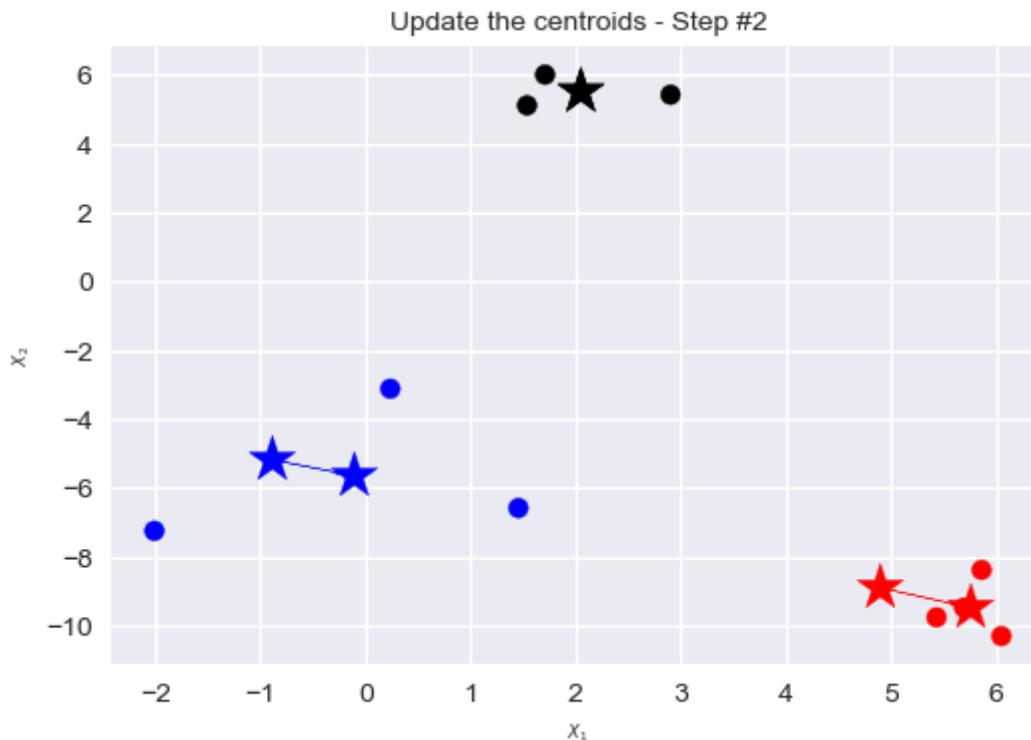
Iteration 3: step 2

- Estimate new centers as *average* of observations in a cluster.

```
In [169]: 1 new_centers_it3 = update_centers(X, z, new_centers_it2, k)
```

- This is how the centers moved in this iteration.

```
In [170]: 1 plot_update_centroid(toy_df, 6, 4, new_centers_it3, new_centers_it2, di
```



Iteration 4: step 1

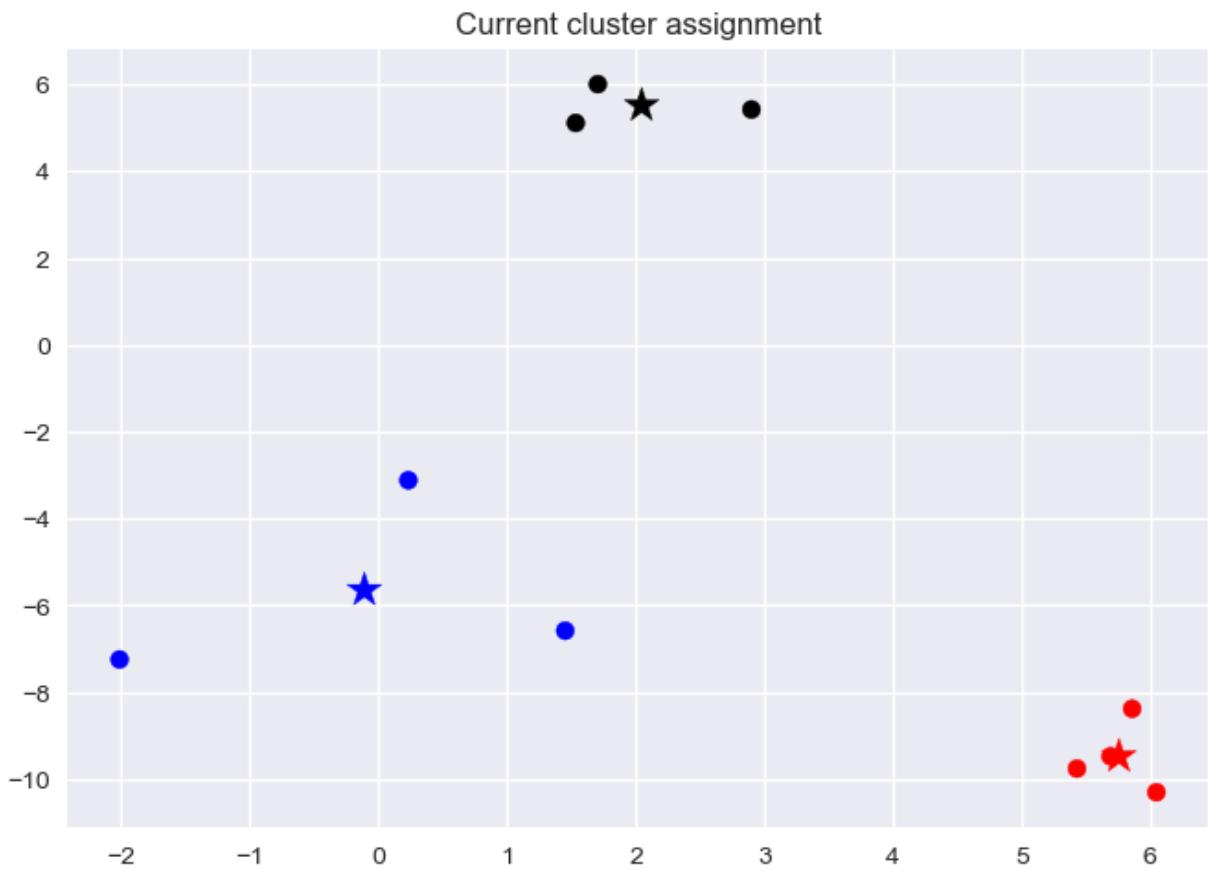
- Assign each example to the closest cluster center.

```
In [171]: 1 dist, z = update_z(X, new_centers_it3)
2 z
```

```
Out[171]: array([2, 0, 1, 0, 2, 2, 1, 1, 0, 2])
```

- This is the current cluster assignment.

```
In [172]: 1 plot_current_assinment(X, z, new_centers_it3)
```



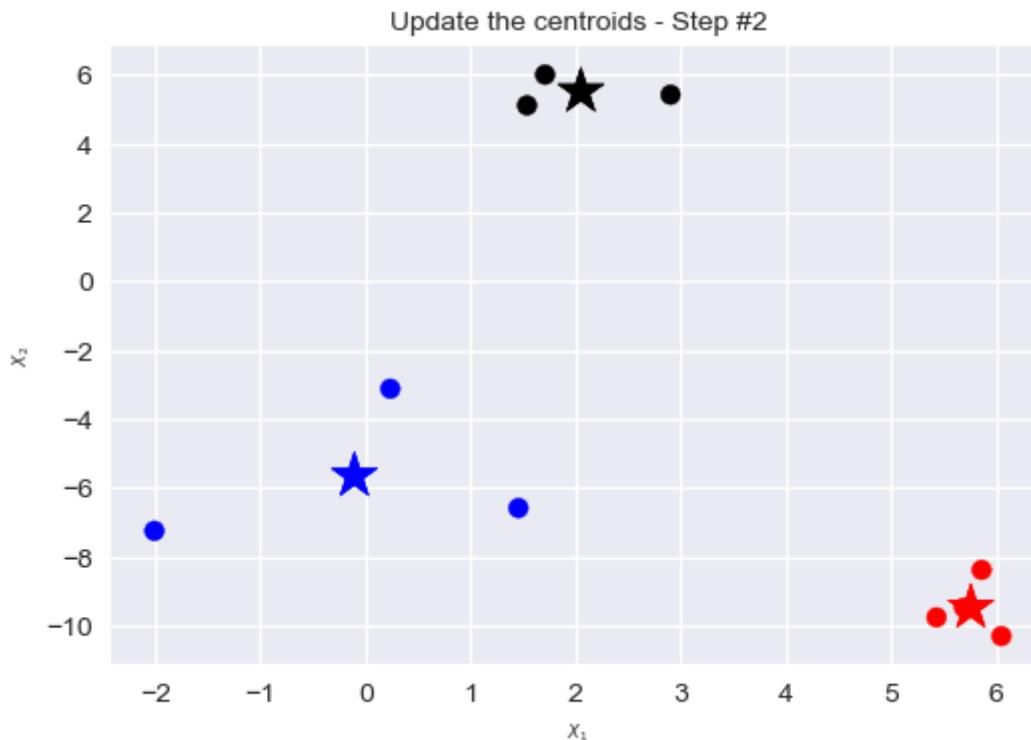
Iteration 4: step 2

- Estimate new centers as *average* of observations in a cluster.

```
In [173]: 1 new_centers_it4 = update_centers(X, z, new_centers_it3, k)
```

- The cluster centers are not moving anymore.

```
In [174]: 1 plot_update_centroid(toy_df, 6, 4, new_centers_it4, new_centers_it3, di
```



Iteration 5: step 1

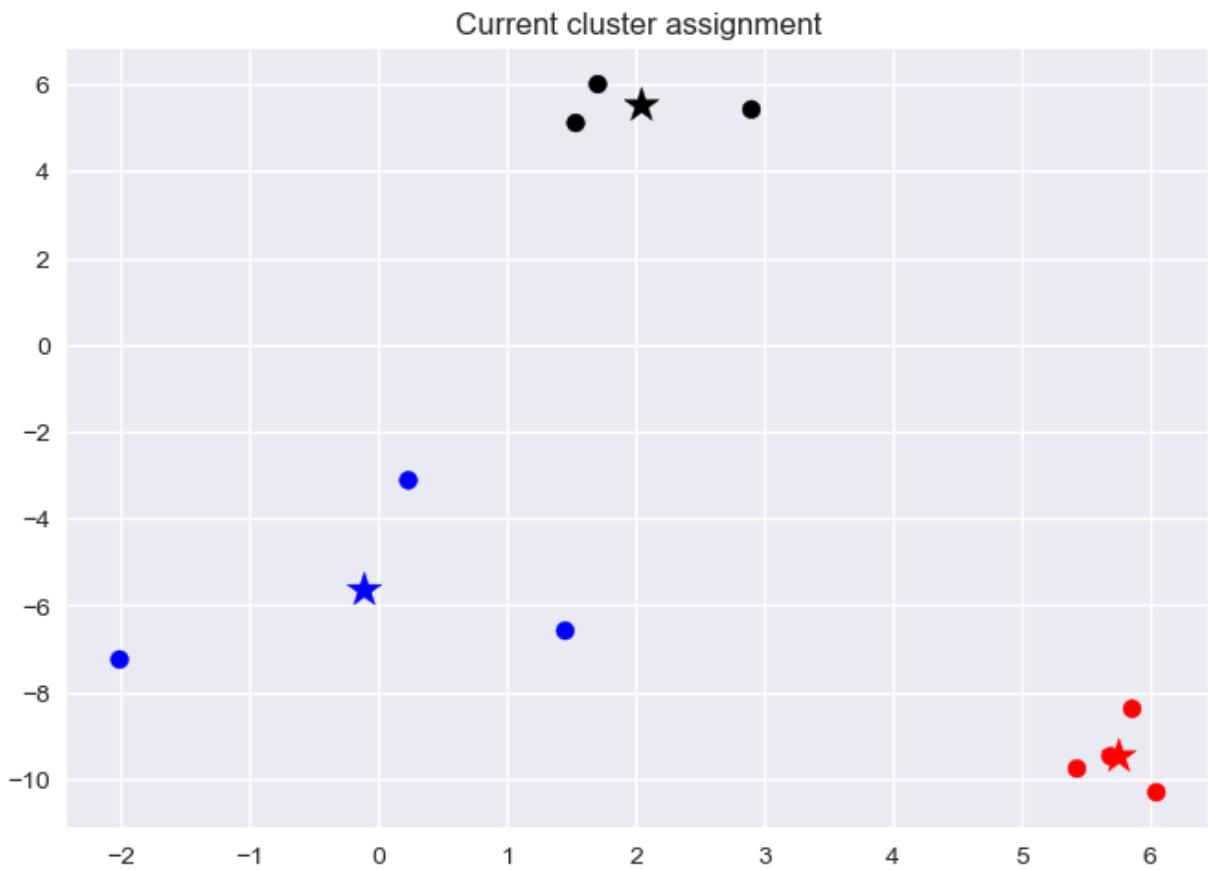
- Assign each example to the closest cluster center.

```
In [175]: 1 dist, z = update_z(X, new_centers_it4)
2 z
```

```
Out[175]: array([2, 0, 1, 0, 2, 2, 1, 1, 0, 2])
```

- This is the current cluster assignment.

```
In [176]: 1 plot_current_assinment(X, z, new_centers_it4)
```



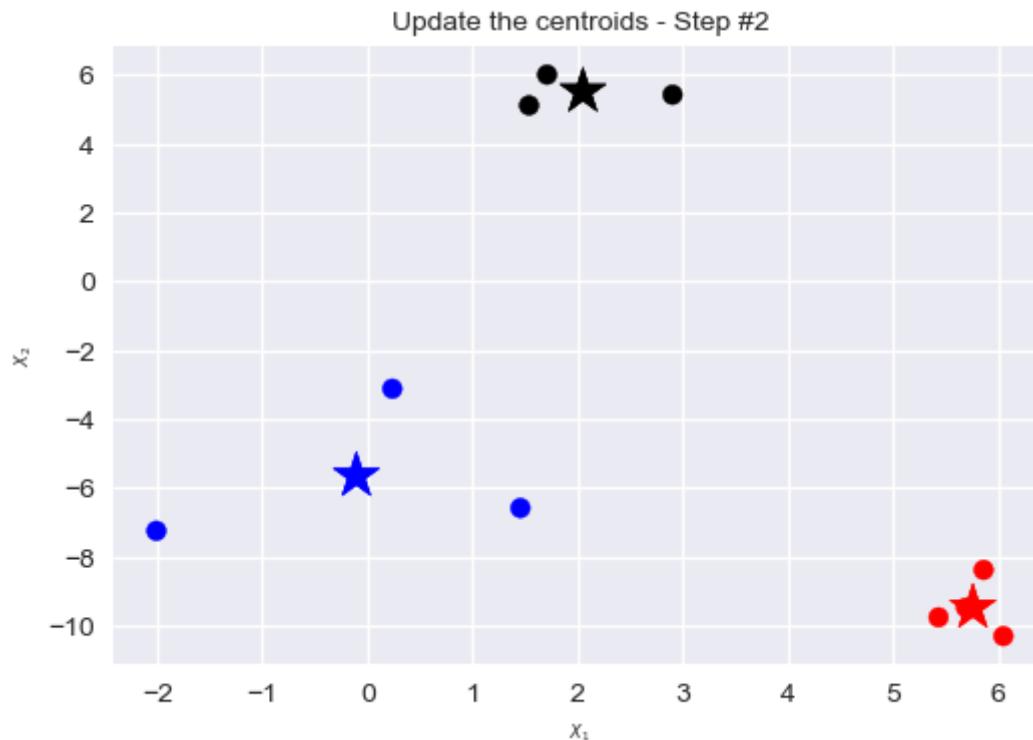
Iteration 5: step 2

- Estimate new centers as *average* of observations in a cluster.

```
In [177]: 1 new_centers_it5 = update_centers(X, z, new_centers_it4, k)
```

- The cluster centers are not moving anymore.

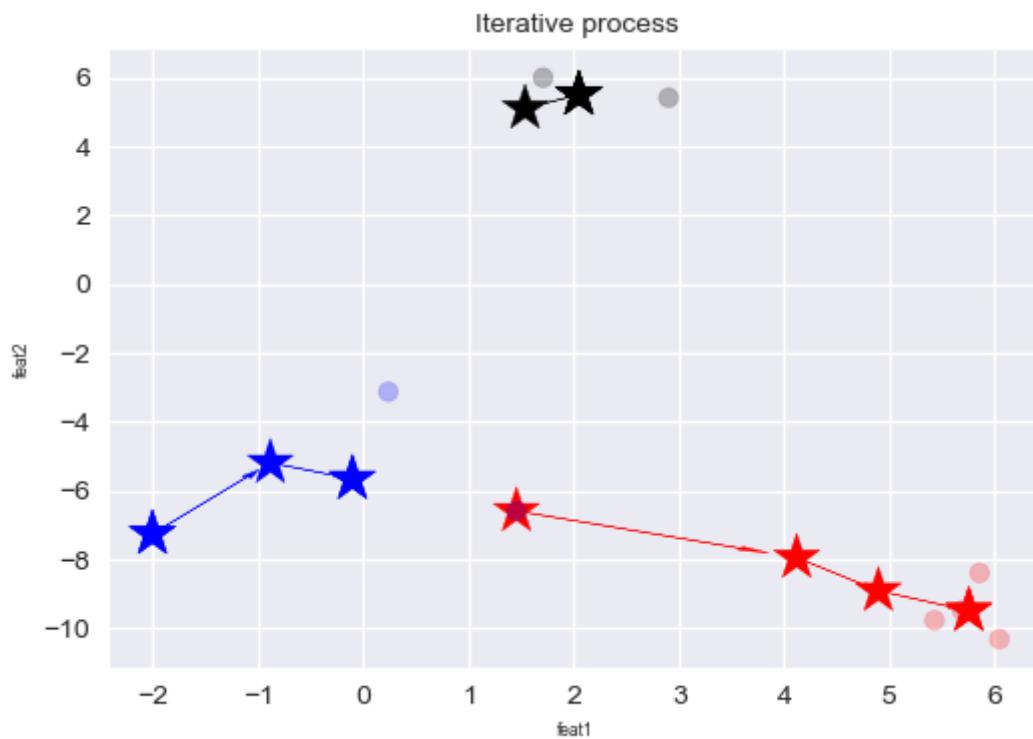
```
In [178]: 1 plot_update_centroid(toy_df, 6, 4, new_centers_it5, new_centers_it4, di
```



When to stop?

- Seems like our centroids aren't changing anymore.
- The algorithm has converged. So we stop!
- K-Means always converges. It doesn't mean it finds the "right" clusters. It can converge to a sub-optimal solution.

```
In [179]: 1 plot_iterative(toy_df, 6, 4, centers)
```



Initialization is crucial. We'll talk about it in a bit.

Example 2

- Let's use the K-means on the iris dataset.

In [180]:

```

1 ## Iris dataset
2 iris = datasets.load_iris() # loading the iris dataset
3 features = iris.data # get the input data
4 labels = iris.target_names[
5     iris.target
6 ] # get the targets, in this case the types of the Iris flower
7
8 iris_df = pd.DataFrame(features, columns=iris.feature_names)
9 iris_df

```

Out[180]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [181]:

```
1 np.unique(labels)
```

Out[181]: array(['setosa', 'versicolor', 'virginica'], dtype='<U10')

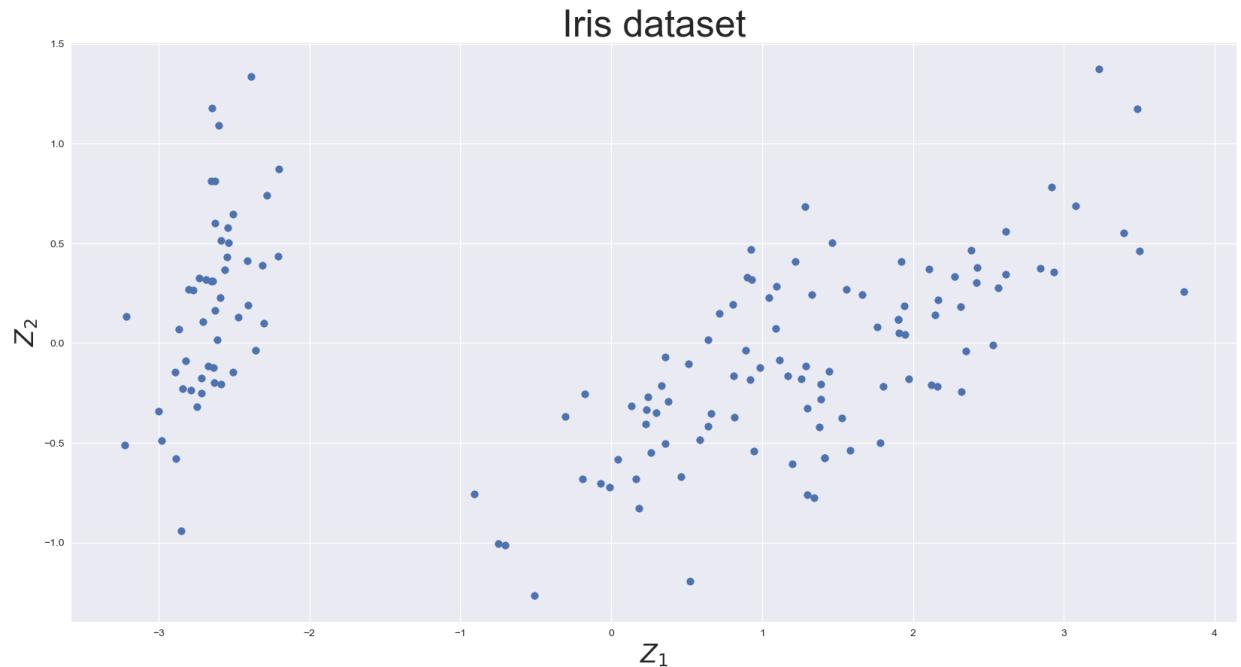
In [182]:

```

1 # Reducing the dimensionality for plotting purposes
2 # (We're going to learn more about it later in the course)
3 pca = PCA(n_components=2)
4 pca.fit(features)
5 data_iris = pd.DataFrame(pca.transform(features), columns=["$z_1$", "$z_2$"])
6 data_iris["target"] = labels

```

```
In [183]: 1 plot_unsup(data_iris, 20, 10, "Iris dataset")
```

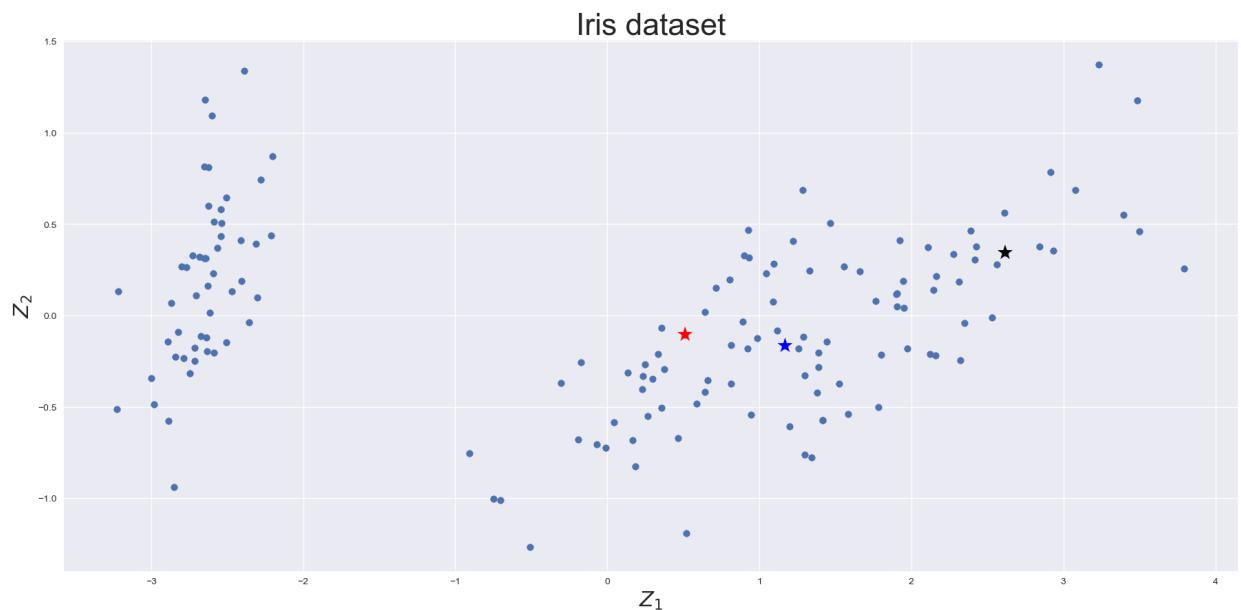


Initialization

- In this case, we know that $k = 3$;
- We are going to pick three points at random to use as initial centroids;

```
In [184]: 1 # RANDOM initialization
```

```
2 k = 3
3 centroids = np.random.choice(range(0, 150), size=k)
4 centroids = data_iris.iloc[centroids, 0:2]
5 plot_intial_center(data_iris, centroids, 22, 10, title="Iris dataset")
```



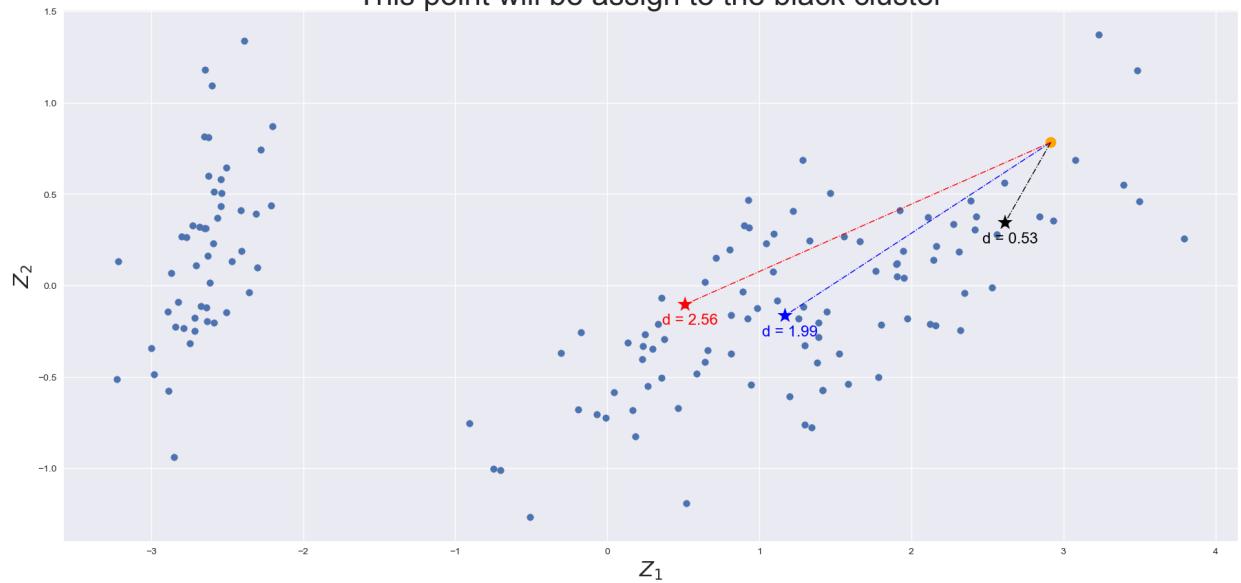
- Next, for each point in the dataset, we calculate the distance to each one of the centroids;

- Let's do it for one point as example:

In [185]:

```
1 plot_example_dist(data_iris, centroids, 22, 10)
```

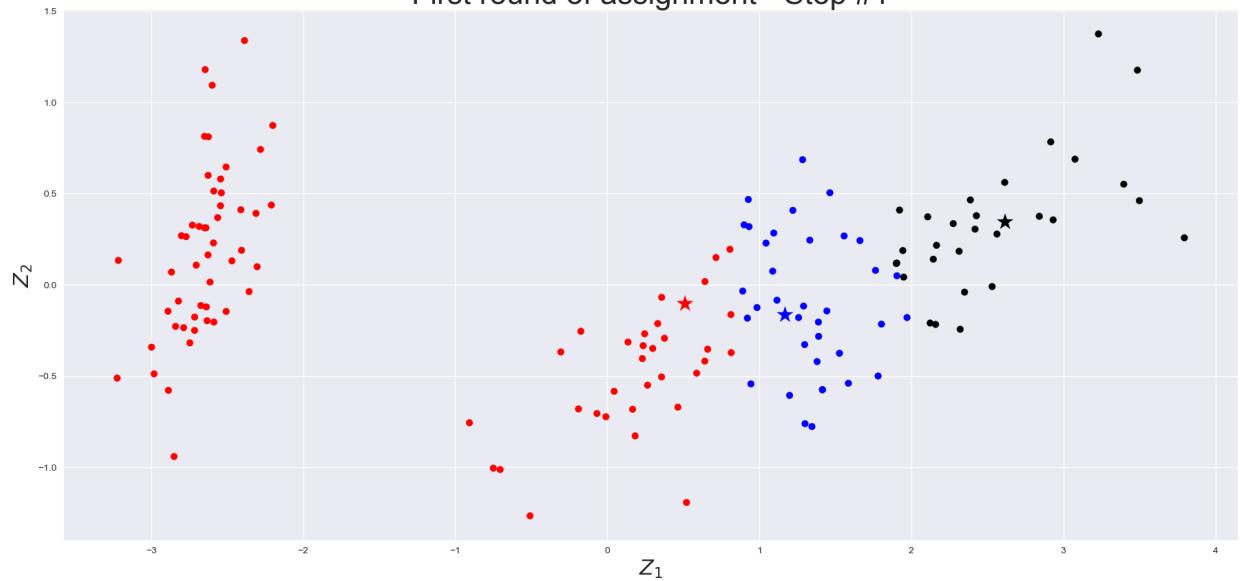
This point will be assigned to the black cluster



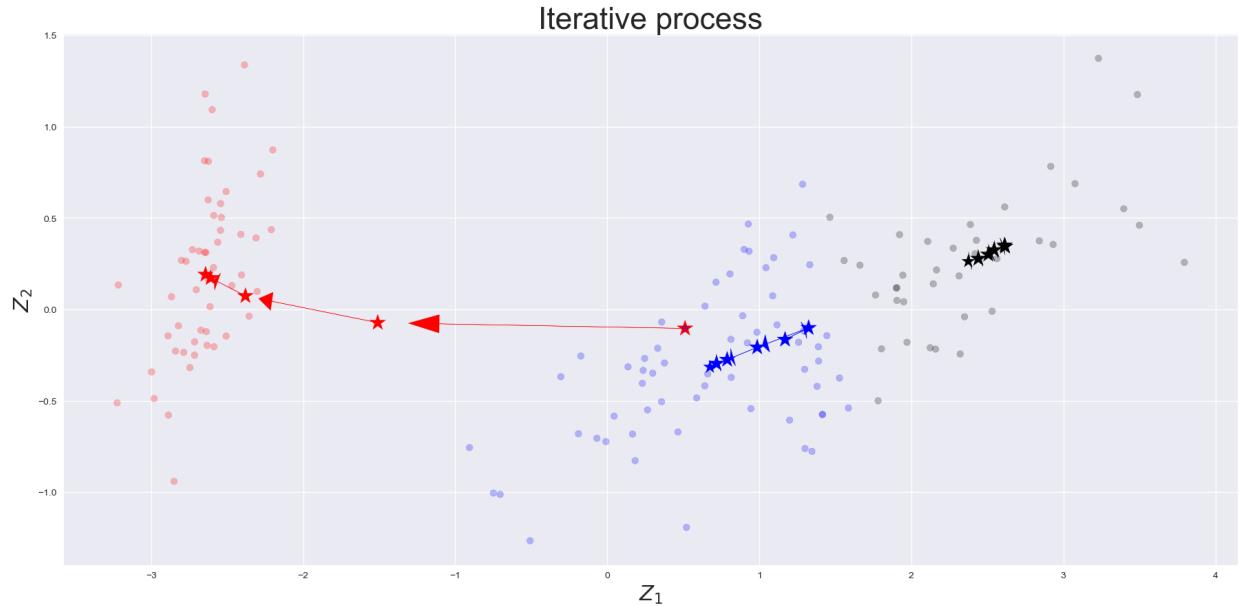
In [186]:

```
1 dist = distance.cdist(data_iris.iloc[:, 0:2], centroids.iloc[:, 0:2])
2 plot_first_assignment(data_iris, centroids, dist, 22, 10)
```

First round of assignment - Step #1



```
In [187]: 1 plot_iterative(data_iris, 22, 10, centroids.to_numpy())
```



(Optional) Feature engineering using K-Means

- K-Means could be used for feature engineering in supervised learning.
- Examples:
 - You could add a categorical feature: cluster membership
 - You could add a continuous features: distance from each cluster center
- See [this paper](http://ai.stanford.edu/~acoates/papers/coatesleeng_aistats_2011.pdf) (http://ai.stanford.edu/~acoates/papers/coatesleeng_aistats_2011.pdf).

Choosing K

Hyperparameter tuning for K

- K-Means takes K (`n_clusters` in `sklearn`) as a hyperparameter. How do we pick K?
- In supervised setting we carried out hyperparameter optimization based on cross-validation scores.
- Since in unsupervised learning we do not have the target values, it becomes difficult to objectively measure the effectiveness of the algorithms.
- There is no definitive approach.
- However, some strategies might be useful to help you determine K.

Method 1: The Elbow method

- This method looks at the sum of **intra-cluster distances**, which is also referred to as **inertia**.
- The intra-cluster distance in our toy example above is given as

$$\sum_{P_i \in C_1} \text{distance}(P_i, C_1)^2 + \sum_{P_i \in C_2} \text{distance}(P_i, C_2)^2 + \sum_{P_i \in C_3} \text{distance}(P_i, C_3)^2$$

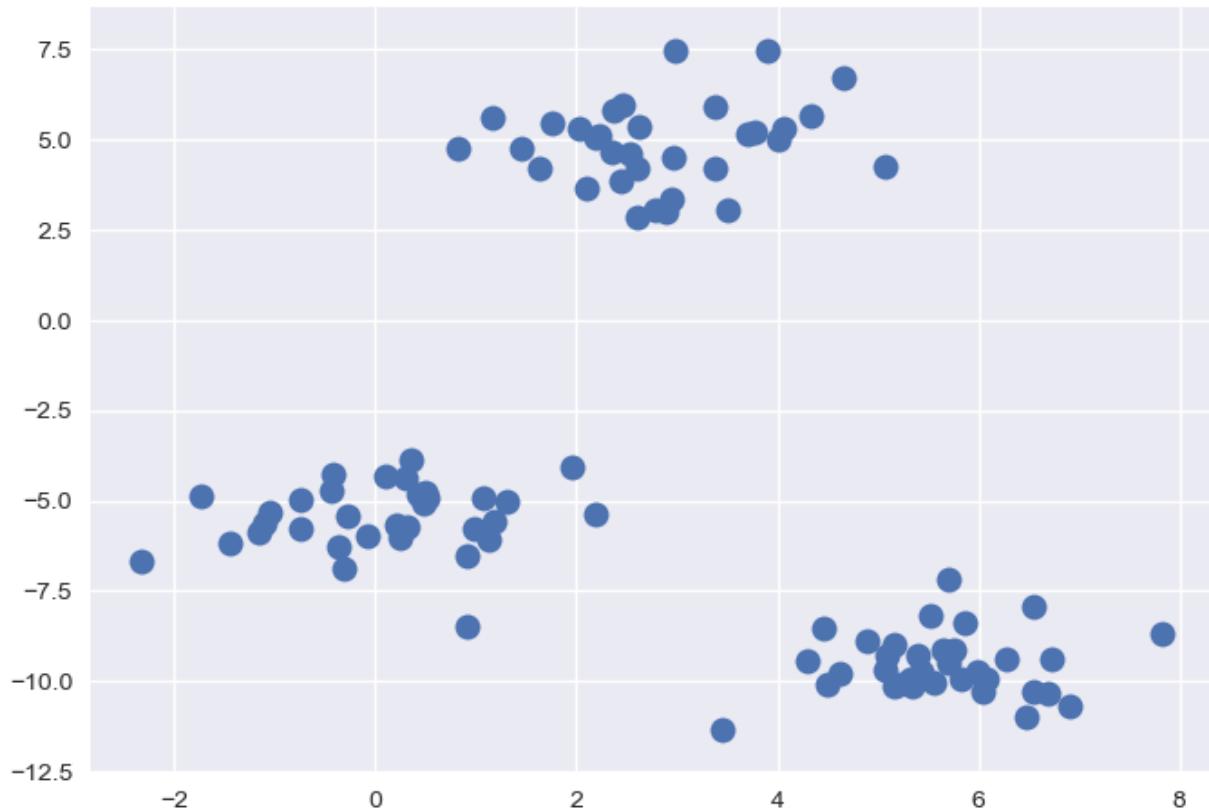
Where

- C_1, C_2, C_3 are centroids
- P_i s are points within that cluster
- *distance* is the usual Euclidean distance.

Inertia

You can access this intra-cluster distance or inertia as follows.

```
In [188]: 1 X, y = make_blobs(centers=3, n_features=2, random_state=10)
2 mlearn.discrete_scatter(X[:, 0], X[:, 1], markers="o");
```



In [189]:

```

1 d = {"K": [], "inertia": []}
2 for k in range(1, 100, 10):
3     model = KMeans(n_clusters=k).fit(X)
4     d["K"].append(k)
5     d["inertia"].append(model.inertia_)

```

In [190]:

```
1 pd.DataFrame(d)
```

Out[190]:

	K	inertia
0	1	4372.460950
1	11	58.474524
2	21	26.900485
3	31	12.770892
4	41	6.421834
5	51	3.593682
6	61	1.961654
7	71	0.945421
8	81	0.322479
9	91	0.053156

- The inertia decreases as K increases.
- Question: Do we want inertia to be small or large?
- The problem is that we can't just look for a k that minimizes inertia because it decreases as k increases.
 - If I have number of clusters = number of examples, each example will have its own cluster and the intra-cluster distance will be 0.
- Instead we evaluate the trade-off: "small k" vs "small intra-cluster distances".

In [191]:

```

1 def plot_elbow(w, h, inertia_values):
2     plt.figure(figsize=(w, h))
3     plt.axvline(x=3, linestyle="--", c="black")
4     plt.plot(range(1, 10), inertia_values, "-o")
5     ax = plt.gca()
6     ax.tick_params("both", labelsize=(w + h) / 2)
7     ax.set_xlabel("K", fontsize=w)
8     ax.set_ylabel("Inertia", fontsize=w)

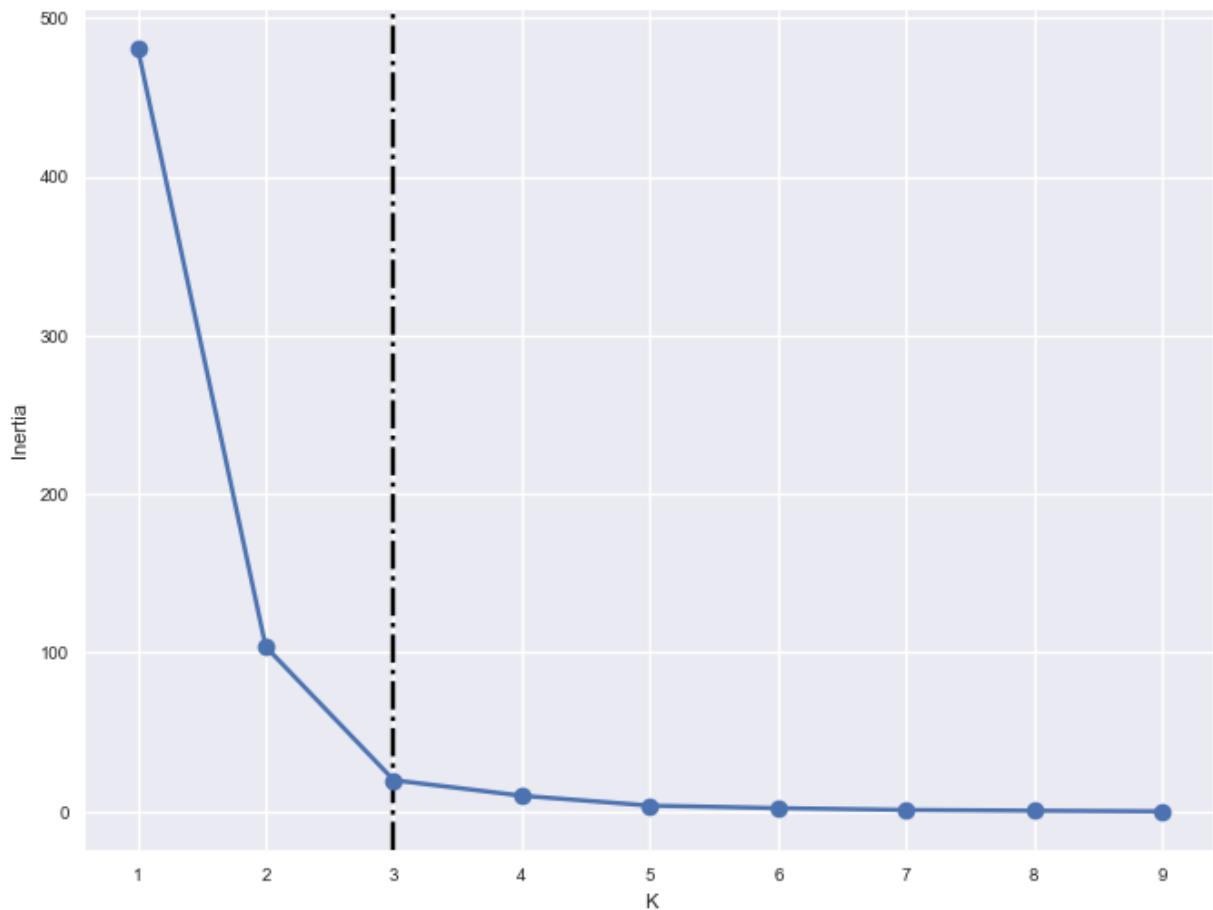
```

In [192]:

```

1 inertia_values = list()
2 for k in range(1, 10):
3     inertia_values.append(KMeans(n_clusters=k).fit(toy_df).inertia_)
4 plot_elbow(8, 6, inertia_values)

```



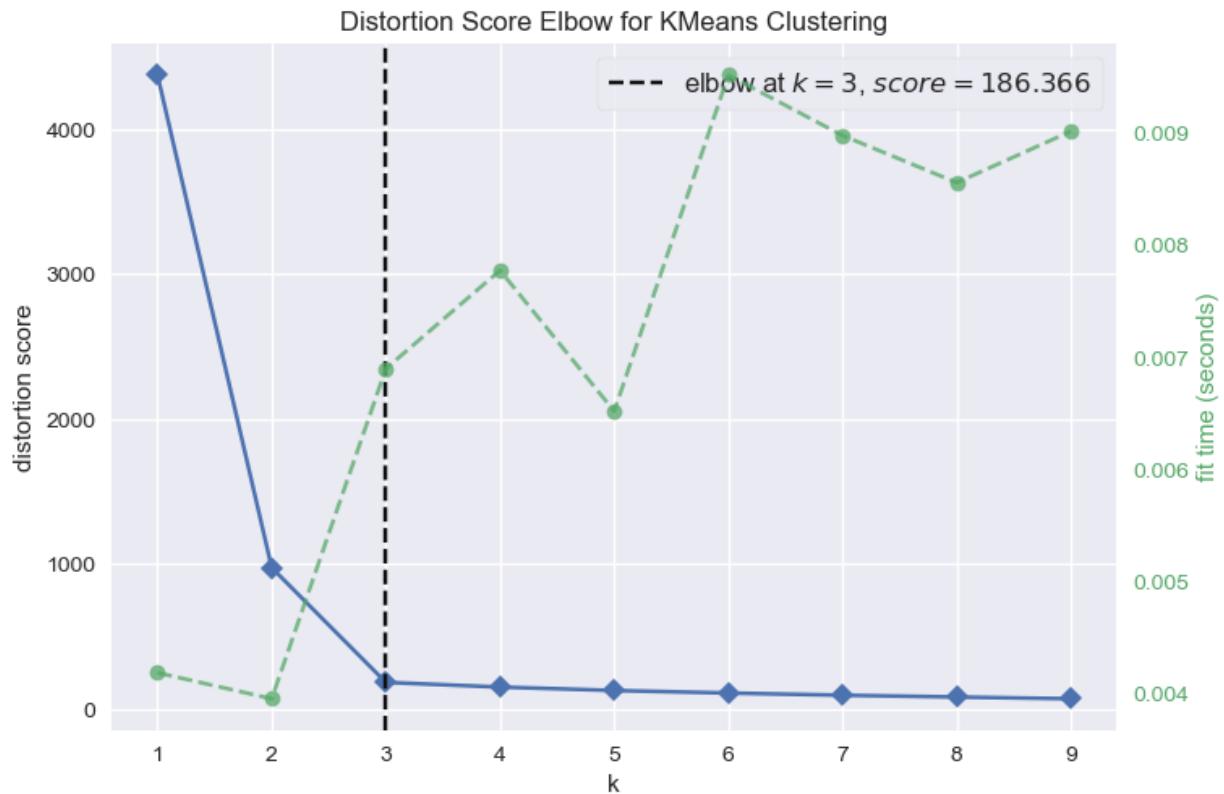
- From the above plot, we could argue that three clusters (the point of inflection on the curve) are enough.
- The inertia decreases when clusters are greater than 3. However it's not a big improvement and so we prefer K=3.
- In this toy example, it's the plot is kind of clear and easy to interpret but it can be hard to interpret in real life examples.

There is a package called `yellowbrick` (<https://www.scikit-yb.org/en/latest/api/cluster/elbow.html>) which can be used to create these plots conveniently.

```
conda install -c districtdatalabs yellowbrick
```

In [193]:

```
1 from yellowbrick.cluster import KElbowVisualizer
2
3 model = KMeans()
4 visualizer = KElbowVisualizer(model, k=(1, 10))
5
6 visualizer.fit(X) # Fit the data to the visualizer
7 visualizer.show();
```

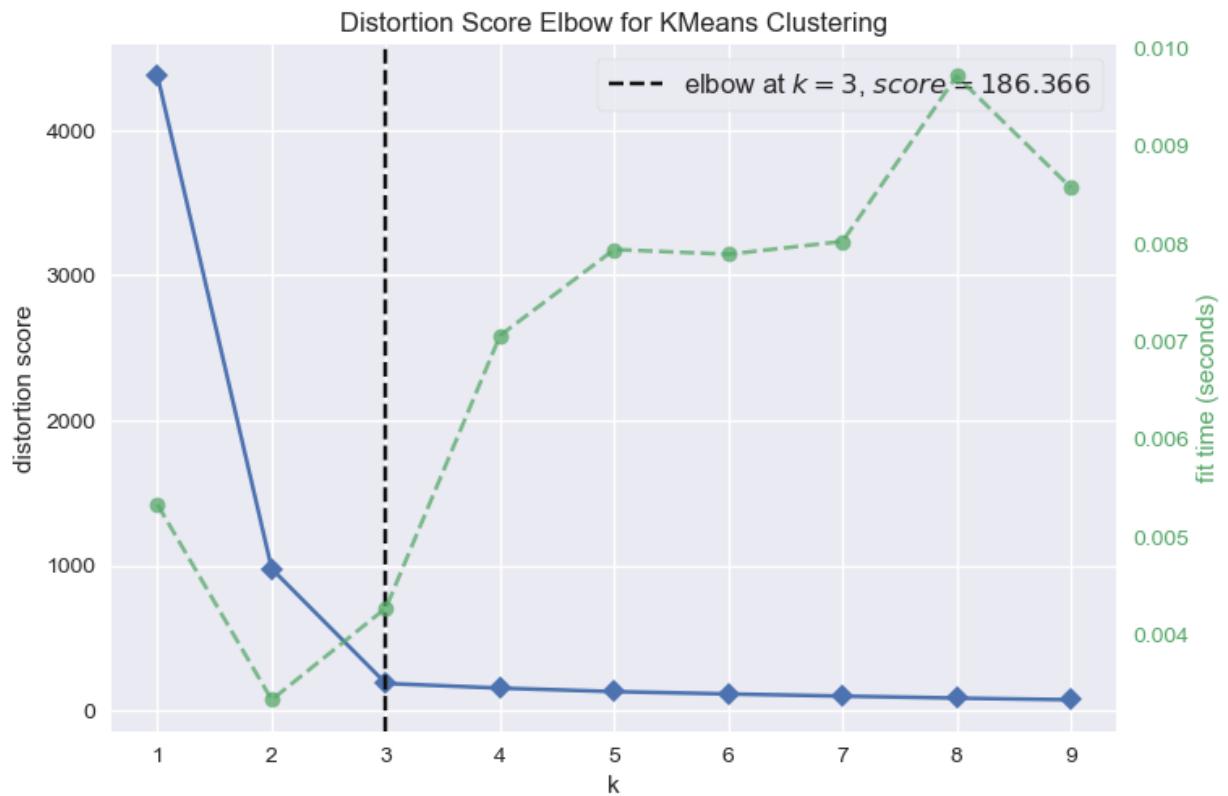


In [194]:

```

1 visualizer = KElbowVisualizer(model, k=(1, 10))
2
3 visualizer.fit(X) # Fit the data to the visualizer
4 visualizer.finalize();

```



In [195]:

```
1 visualizer.show();
```

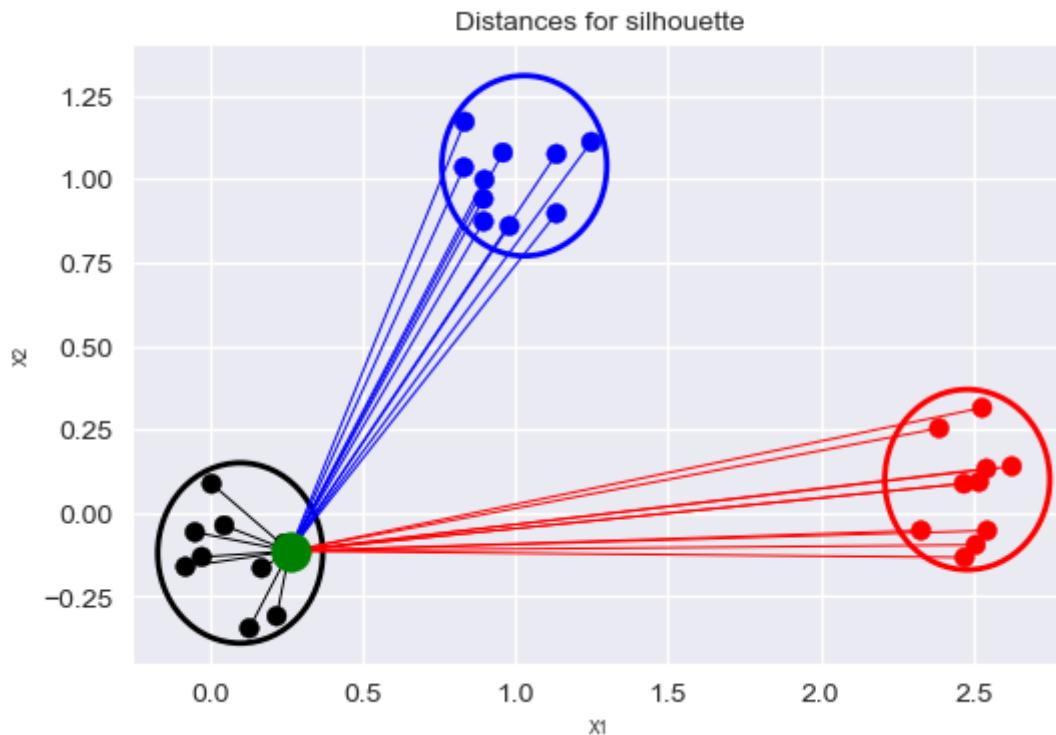
Method 2: The Silhouette method

- Not dependent on the notion of cluster centers.
- Calculated using the **mean intra-cluster distance** (a) and the **mean nearest-cluster distance** (b) for each sample.

Mean intra-cluster distance (a)

- Suppose the green point below is our sample.
- Average of the distances of the green point to the other points in the same cluster.
 - These distances are represented by the black lines.

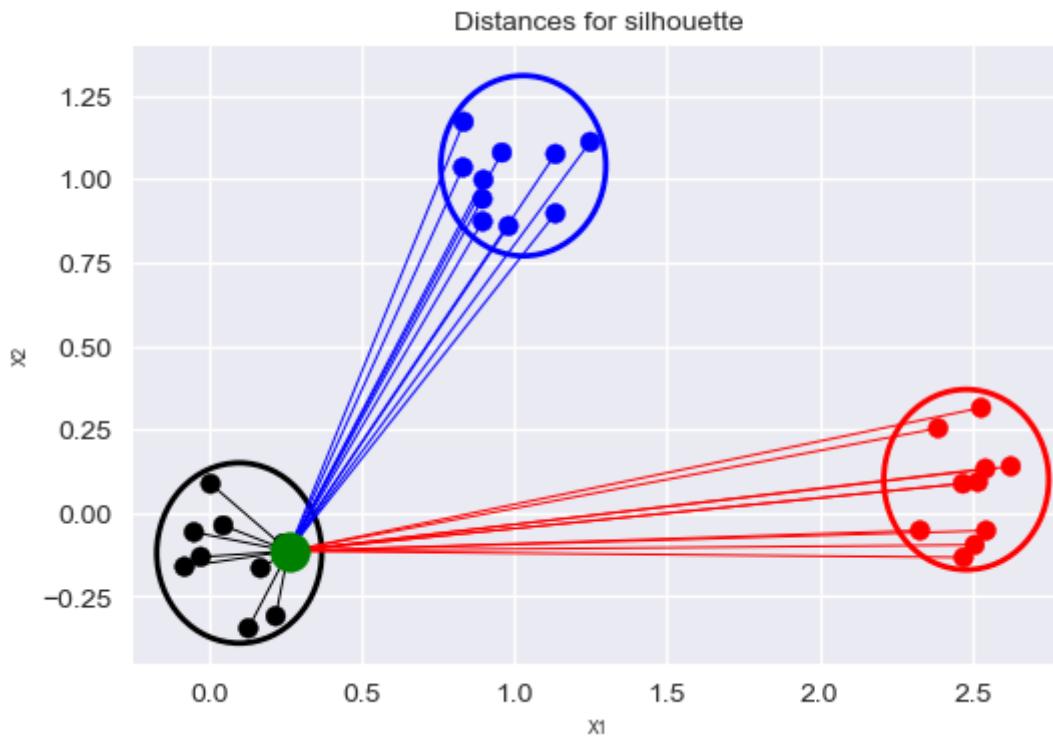
In [196]: 1 plot_silhouette_dist(6, 4)



Mean nearest-cluster distance (b)

- Average of the distances of the green point to the blue points is smaller than the average of the distances of the green point to the red points. So the **nearest cluster** is the blue cluster.
- So the mean nearest-cluster distance is the average of the distances of the green point to the blue points.

In [197]: 1 plot_silhouette_dist(6, 4)



Silhouette distance for a sample

- the difference between the **average nearest-cluster distance** (b) and **average intra-cluster distance** (a) for each sample, normalized by the maximum value

$$\frac{b - a}{\max(a, b)}$$

- The best value is 1.
- The worst value is -1 (samples have been assigned to wrong clusters).
- Value near 0 means overlapping clusters.

The overall **Silhouette score** is the average of the Silhouette scores for all samples.

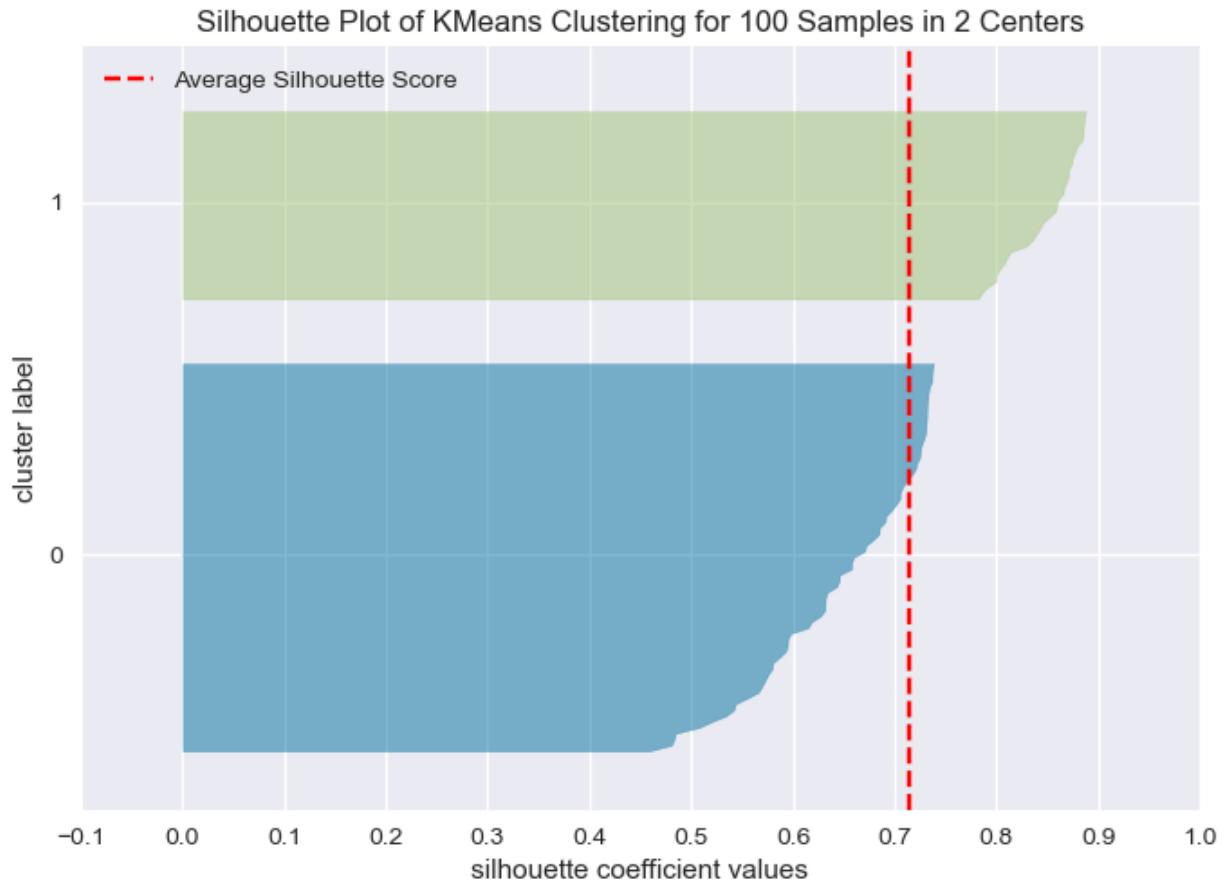
Using Silhouette scores to select the number of clusters

- The plots below show the Silhouette scores for each sample in that cluster.
- Higher values indicate well-separated clusters.
- The size of the Silhouette shows the number of samples and hence shows imbalance of data points in clusters.

In [198]: 1 `from yellowbrick.cluster import SilhouetteVisualizer`

In [199]:

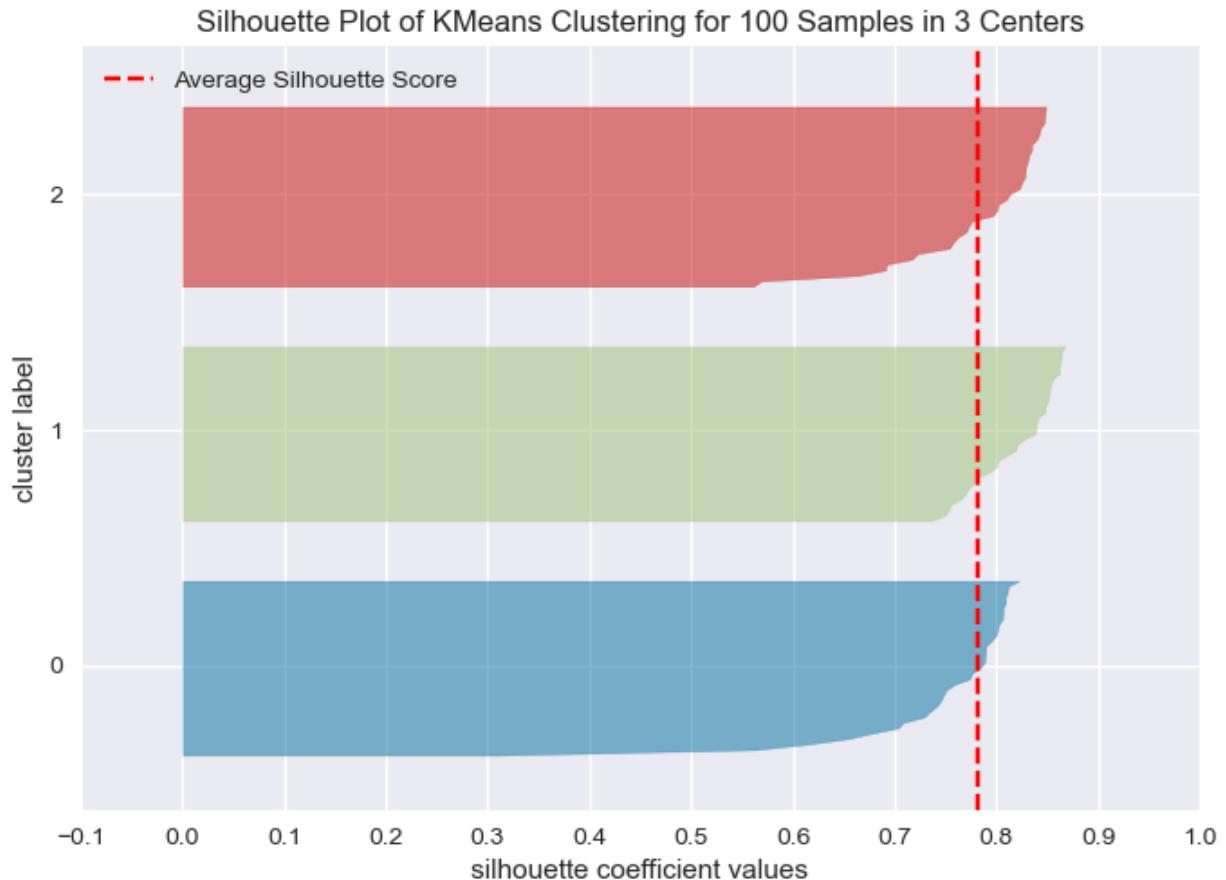
```
1 model = KMeans(2, random_state=42)
2 visualizer = SilhouetteVisualizer(model, colors="yellowbrick")
3 visualizer.fit(X) # Fit the data to the visualizer
4 visualizer.show() # Finalize and render the figure
```



Out[199]: <AxesSubplot: title={'center': 'Silhouette Plot of KMeans Clustering for 100 Samples in 2 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

In [200]:

```
1 model = KMeans(3, random_state=42)
2 visualizer = SilhouetteVisualizer(model, colors="yellowbrick")
3 visualizer.fit(X) # Fit the data to the visualizer
4 visualizer.show() # Finalize and render the figure
```



Out[200]: <AxesSubplot: title={'center': 'Silhouette Plot of KMeans Clustering for 100 Samples in 3 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

In [201]:

```

1 model = KMeans(5, random_state=42)
2 visualizer = SilhouetteVisualizer(model, colors="yellowbrick")
3 visualizer.fit(X) # Fit the data to the visualizer
4 visualizer.show() # Finalize and render the figure
5

```



Out[201]: <AxesSubplot: title={'center': 'Silhouette Plot of KMeans Clustering for 100 Samples in 5 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

What to look for in these plots?

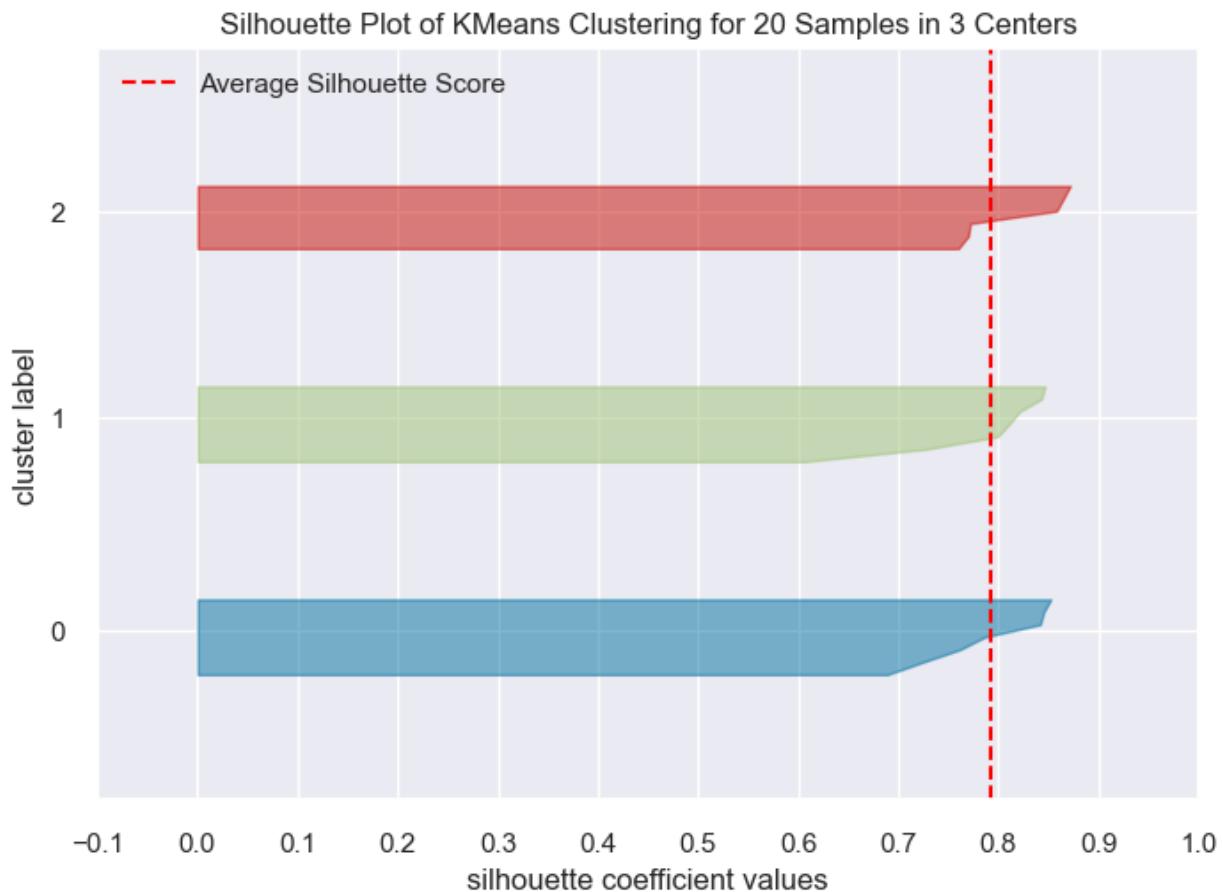
- Unlike inertia, larger values are better because they indicate that the point is further away from neighbouring clusters.
- The thickness of each silhouette indicates the cluster size.
- The shape of each silhouette indicates the "goodness" for points in each cluster.
- The length (or area) of each silhouette indicates the goodness of each cluster.
- A slower dropoff (more rectangular) indicates more points are "happy" in their cluster.

In [238]:

```

1 model = KMeans(3, random_state=42)
2 visualizer = SilhouetteVisualizer(model, colors="yellowbrick")
3 visualizer.fit(X) # Fit the data to the visualizer
4 visualizer.show() # Finalize and render the figure
5

```



Out[238]: <AxesSubplot: title={'center': 'Silhouette Plot of KMeans Clustering for 20 Samples in 3 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

Comments on Silhouette scores

- Unlike inertia, larger values are better because they indicate that the point is further away from neighbouring clusters.
- Unlike inertia, the overall silhouette score gets worse as you add more clusters because you end up being closer to neighbouring clusters.
- Thus, as with inertia, you will not see a "peak" value of this metric that indicates the best number of clusters.
- We can visualize the silhouette score for each example individually in a silhouette plot (hence the name), see below.
- We can apply Silhouette method to clustering methods other than K-Means.

K-Means: True/False questions

- When choosing a number of clusters, we want to minimize inertia.
- K-Means algorithm always converges to the same solution.
- In some iterations some points may be left unassigned.
- K-means terminates when the centroid locations do not change between iterations.
- It is possible to have negative silhouette score values.

K-Means case study: Customer segmentation

What is customer segmentation?

Check out [this interesting talk by Malcom Gladwell](#) (https://www.ted.com/talks/malcolm_gladwell_on_spaghetti_sauce?language=en). Humans are diverse and there is no single spaghetti sauce that would make all of them happy!

Often it's beneficial to businesses to explore the landscape of the market and tailor their services and products offered to each group. This is called **customer segmentation**. It's usually applied when the dataset contains some of the following features.

- **Demographic information** such as gender, age, marital status, income, education, and occupation
- **Geographical information** such as specific towns or counties or a customer's city, state, or even country of residence (in case of big global companies)
- **Psychographics** such as social class, lifestyle, and personality traits
- **Behavioral data** such as spending and consumption habits, product/service usage, and desired benefits

Business problem

- Imagine that you are hired as a data scientist at a bank. They provide some data of their credit card customers to you.
- Their goal is to develop customized marketing campaigns and they ask you to group customers based on the given information.
- Now that you know about K-Means clustering, let's apply it to the dataset to group customers.

Data

- We will use the [Credit Card Dataset for clustering](#) (<https://www.kaggle.com/arjunbhavin2013/ccdata>) from Kaggle.
- Download the data and save the CSV under the `data` folder.
- I encourage you to work through this case study on your own.

```
In [203]: 1 creditcard_df = pd.read_csv("../data/CC_General.csv")
2 creditcard_df.shape
```

Out[203]: (8950, 18)

Information of the dataset

We have behavioral data.

- CUSTID: Identification of Credit Card holder
- BALANCE: Balance amount left in customer's account to make purchases
- BALANCE_FREQUENCY: How frequently the Balance is updated, score between 0 and 1 (1 = frequently updated, 0 = not frequently updated)
- PURCHASES: Amount of purchases made from account
- ONEOFFPURCHASES: Maximum purchase amount done in one-go
- INSTALLMENTS_PURCHASES: Amount of purchase done in installment
- CASH_ADVANCE: Cash in advance given by the user
- PURCHASES_FREQUENCY: How frequently the Purchases are being made, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased)
- ONEOFF_PURCHASES_FREQUENCY: How frequently Purchases are happening in one-go (1 = frequently purchased, 0 = not frequently purchased)
- PURCHASES_INSTALLMENTS_FREQUENCY: How frequently purchases in installments are being done (1 = frequently done, 0 = not frequently done)
- CASH_ADVANCE_FREQUENCY: How frequently the cash in advance being paid
- CASH_ADVANCE_TRX: Number of Transactions made with "Cash in Advance"
- PURCHASES_TRX: Number of purchase transactions made
- CREDIT_LIMIT: Limit of Credit Card for user
- PAYMENTS: Amount of Payment done by user
- MINIMUM_PAYMENTS: Minimum amount of payments made by user
- PRC_FULL_PAYMENT: Percent of full payment paid by user
- TENURE: Tenure of credit card service for user

Preliminary EDA

In [204]: 1 creditcard_df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   CUST_ID          8950 non-null    object  
 1   BALANCE          8950 non-null    float64 
 2   BALANCE_FREQUENCY 8950 non-null    float64 
 3   PURCHASES         8950 non-null    float64 
 4   ONEOFF_PURCHASES 8950 non-null    float64 
 5   INSTALLMENTS_PURCHASES 8950 non-null    float64 
 6   CASH_ADVANCE      8950 non-null    float64 
 7   PURCHASES_FREQUENCY 8950 non-null    float64 
 8   ONEOFF_PURCHASES_FREQUENCY 8950 non-null    float64 
 9   PURCHASES_INSTALLMENTS_FREQUENCY 8950 non-null    float64 
 10  CASH_ADVANCE_FREQUENCY 8950 non-null    float64 
 11  CASH_ADVANCE_TRX 8950 non-null    int64  
 12  PURCHASES_TRX     8950 non-null    int64  
 13  CREDIT_LIMIT      8949 non-null    float64 
 14  PAYMENTS          8950 non-null    float64 
 15  MINIMUM_PAYMENTS 8637 non-null    float64 
 16  PRC_FULL_PAYMENT 8950 non-null    float64 
 17  TENURE            8950 non-null    int64  
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

- All numeric features
- Some missing values

In [205]: 1 creditcard_df.describe()

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_F
count	8950.000000	8950.000000	8950.000000	8950.000000	
mean	1564.474828	0.877271	1003.204834	592.437371	
std	2081.531879	0.236904	2136.634782	1659.887917	
min	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	
75%	2054.140036	1.000000	1110.130000	577.405000	
max	19043.138560	1.000000	49039.570000	40761.250000	2

Practice exercises for you

1. What is the average `BALANCE` amount?
2. How often the `BALANCE_FREQUENCY` is updated on average?
3. Obtain the row the customer who made the maximum cash advance transaction.

Mini exercises for you (Answers)

1. What is the average `BALANCE` amount? 1564.47
2. How often the `BALANCE_FREQUENCY` is updated on average? 0.88 (pretty often)
3. Obtain the row of the customer who made the maximum cash advance transaction.

```
In [206]: 1 max_cash_advance = creditcard_df[ "CASH_ADVANCE" ].max( )  
2 creditcard_df[creditcard_df[ "CASH_ADVANCE" ] == max_cash_advance]
```

```
Out[206]:      CUST_ID    BALANCE  BALANCE_FREQUENCY  PURCHASES  ONEOFF_PURCHASES  INSTALLI  
2159     C12226  10905.05381           1.0        431.93          133.5
```

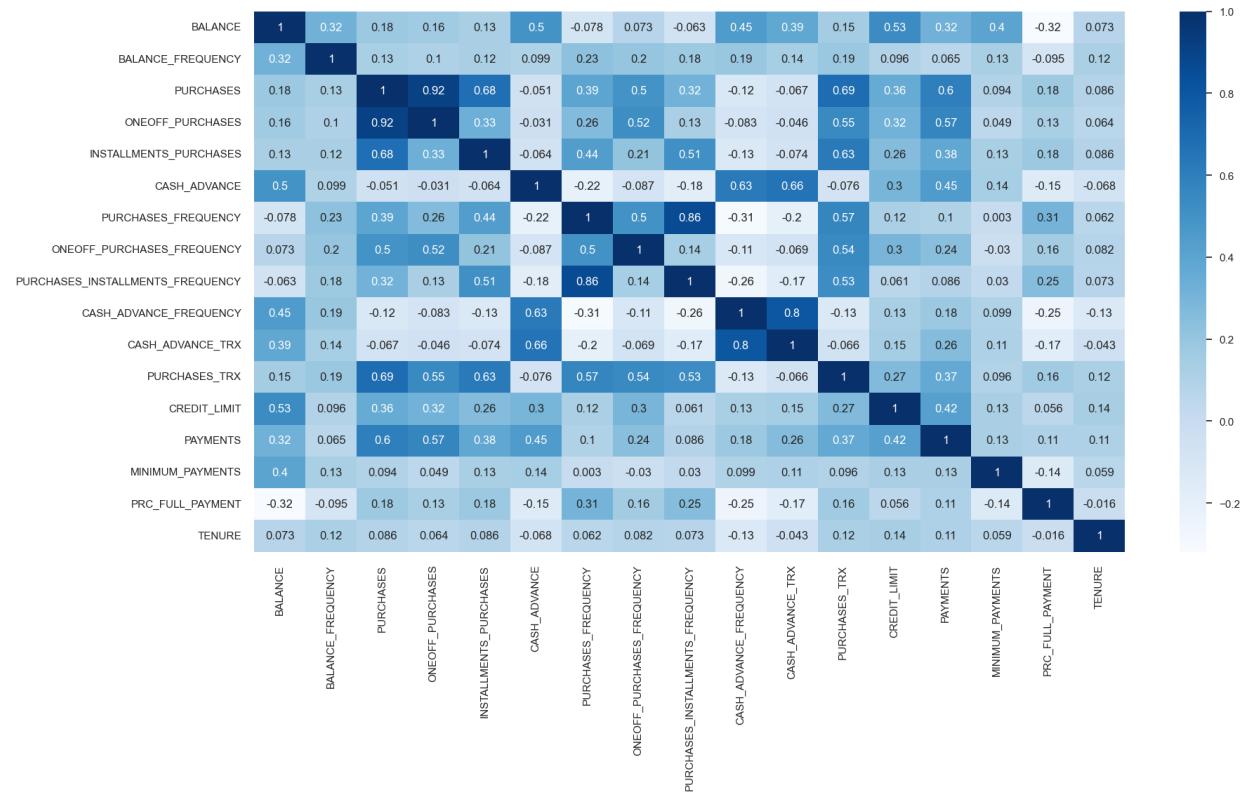
Let's examine correlations between features.

In [207]:

```

1 cor = creditcard_df.corr(numeric_only=True)
2 plt.figure(figsize=(20, 10))
3 sns.set(font_scale=1)
4 sns.heatmap(cor, annot=True, cmap=plt.cm.Blues);

```



Feature types and preprocessing

Let's identify different feature types and transformations

In [208]:

```
1 creditcard_df.columns
```

Out[208]:

```
Index(['CUST_ID', 'BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES',
       'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE',
       'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY',
       'PURCHASES_INSTALLMENTS_FREQUENCY', 'CASH_ADVANCE_FREQUENCY',
       'CASH_ADVANCE_TRX', 'PURCHASES_TRX', 'CREDIT_LIMIT', 'PAYMENTS',
       'MINIMUM_PAYMENTS', 'PRC_FULL_PAYMENT', 'TENURE'],
      dtype='object')
```

In [209]:

```

1 drop_features = ["CUST_ID"]
2 numeric_features = list(set(creditcard_df.columns) - set(drop_features))

```

In [210]:

```

1 from sklearn.impute import SimpleImputer
2
3 numeric_transformer = make_pipeline(SimpleImputer(), StandardScaler())
4
5 preprocessor = make_column_transformer(
6     (numeric_transformer, numeric_features), ("drop", drop_features)
7 )

```

In [211]:

```

1 transformed_df = pd.DataFrame(
2     data=preprocessor.fit_transform(creditcard_df), columns=numeric_fea
3 )

```

In [212]:

```
1 transformed_df
```

Out[212]:

	CASH_ADVANCE	MINIMUM_PAYMENTS	CASH_ADVANCE_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY
0	-0.466786	-3.109675e-01		-0.675349
1	2.605605	8.931021e-02		0.573963
2	-0.466786	-1.016632e-01		-0.675349
3	-0.368653	4.878305e-17		-0.258913
4	-0.466786	-2.657913e-01		-0.675349
...
8945	-0.466786	-3.498541e-01		-0.675349
8946	-0.466786	4.878305e-17		-0.675349
8947	-0.466786	-3.354655e-01		-0.675349
8948	-0.449352	-3.469065e-01		0.157527
8949	-0.406205	-3.329464e-01		0.990398

8950 rows × 17 columns

Now that we have transformed the data, we are ready to run K-Means to cluster credit card customers.

Tuning the hyperparameter `n_clusters`

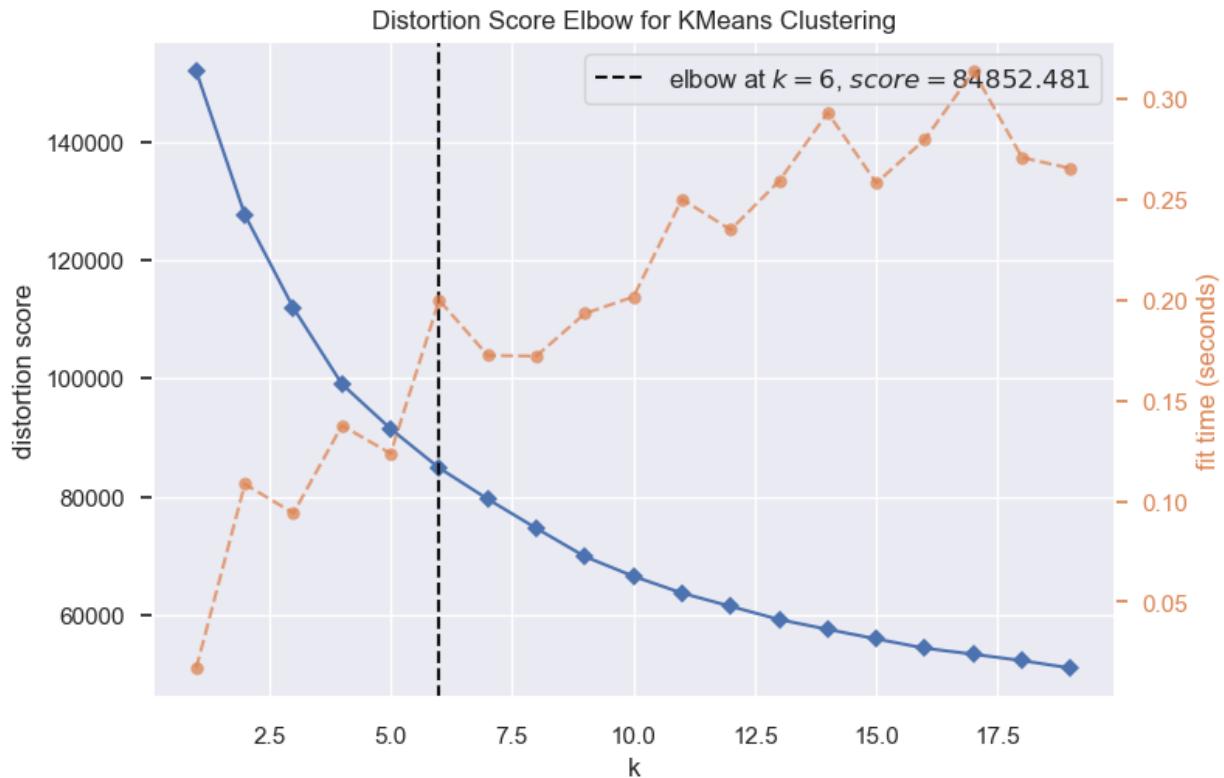
- Let's first obtain optimal number of clusters using the Elbow method.

In [213]:

```

1 model = KMeans()
2 visualizer = KElbowVisualizer(model, k=(1, 20))
3
4 visualizer.fit(transformed_df) # Fit the data to the visualizer
5 visualizer.show();

```



- The optimal number of clusters is not as clear as it was in our toy example.

- Let's examine Silhouette scores.

In [214]:

```
1 model = KMeans(4, random_state=42)
2 visualizer = SilhouetteVisualizer(model, colors="yellowbrick")
3 visualizer.fit(transformed_df) # Fit the data to the visualizer
4 visualizer.show() # Finalize and render the figure
```



Out[214]: <AxesSubplot: title={'center': 'Silhouette Plot of KMeans Clustering for 8950 Samples in 4 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

In [215]:

```
1 model = KMeans(5, random_state=42)
2 visualizer = SilhouetteVisualizer(model, colors="yellowbrick")
3 visualizer.fit(transformed_df) # Fit the data to the visualizer
4 visualizer.show() # Finalize and render the figure
```



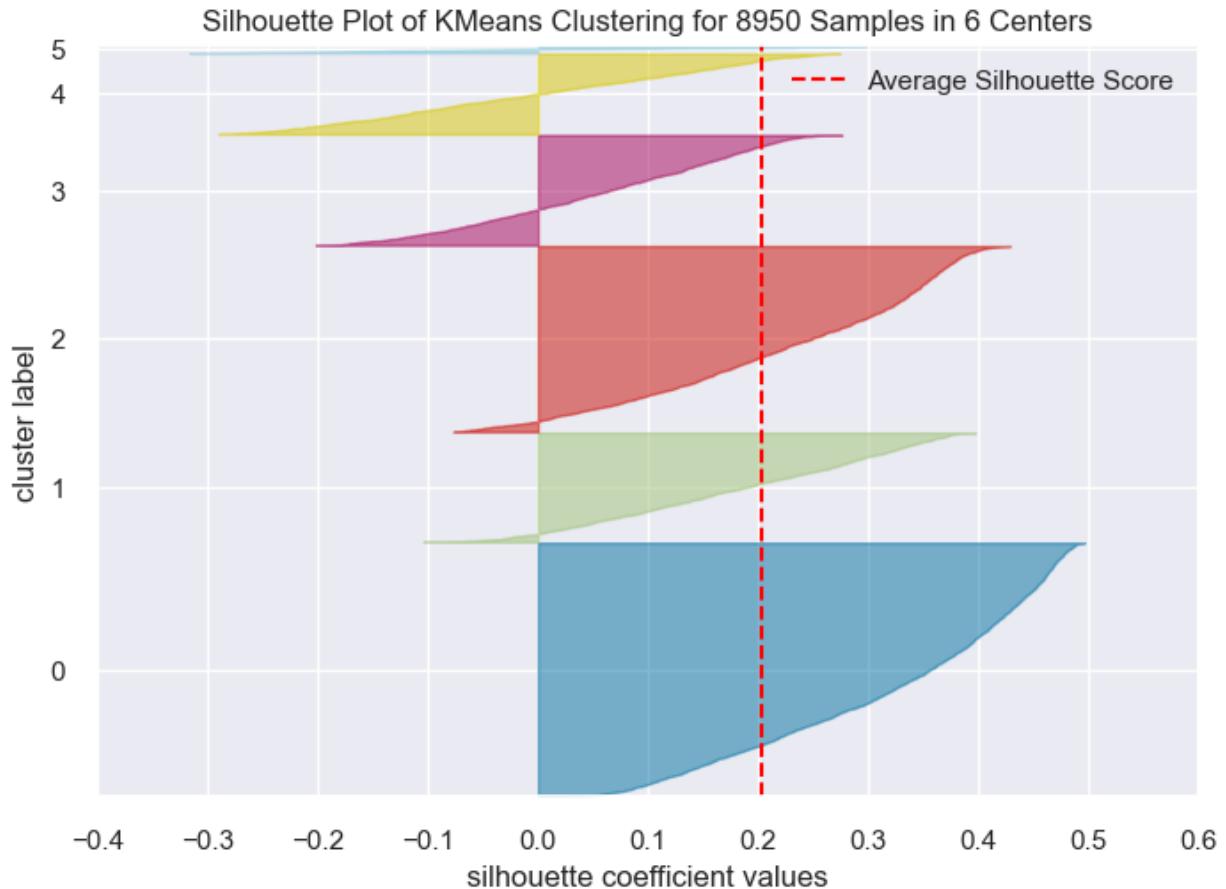
Out[215]: <AxesSubplot: title={'center': 'Silhouette Plot of KMeans Clustering for 8950 Samples in 5 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

In [216]:

```

1 model = KMeans(6, random_state=42)
2 visualizer = SilhouetteVisualizer(model, colors="yellowbrick")
3 visualizer.fit(transformed_df) # Fit the data to the visualizer
4 visualizer.show() # Finalize and render the figure
5

```



Out[216]: <AxesSubplot: title={'center': 'Silhouette Plot of KMeans Clustering for 8950 Samples in 6 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

- I'm going to run KMeans with n_clusters = 4.
- You can try out n_clusters = 5 and n_clusters = 6 on your own.

In [217]:

```

1 kmeans = KMeans(4, random_state=123)
2 kmeans.fit(transformed_df)
3 labels = kmeans.labels_
4 kmeans.cluster_centers_.shape

```

Out[217]: (4, 17)

- Let's visualize the clusters in two dimensions using PCA, which is a popular dimensionality reduction technique.
- We won't be talking about PCA in this course but I'll be using it for visualization.

In [218]:

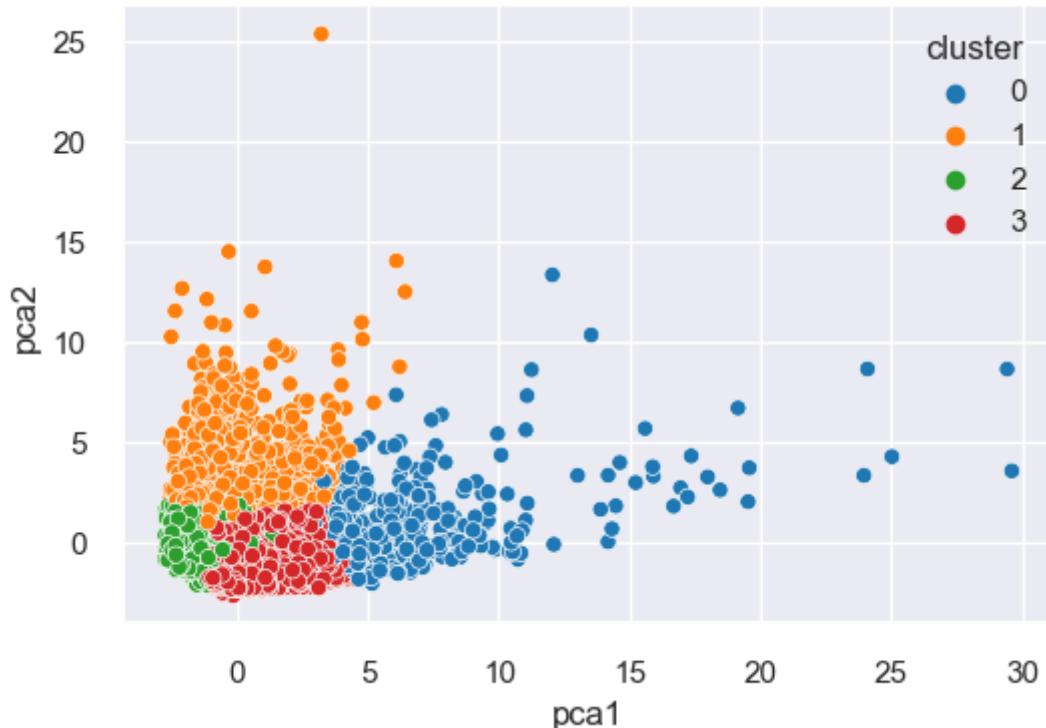
```

1 # Obtain the principal components
2 from sklearn.decomposition import PCA
3
4
5 def plot_pca_clusters(data, labels):
6     """
7         Carries out dimensionality reduction on the data for visualization
8     """
9     pca = PCA(n_components=2)
10    principal_comp = pca.fit_transform(data)
11    pca_df = pd.DataFrame(
12        data=principal_comp, columns=["pca1", "pca2"], index=data.index
13    )
14    pca_df["cluster"] = labels
15    plt.figure(figsize=(6, 4))
16    ax = sns.scatterplot(
17        x="pca1", y="pca2", hue="cluster", data=pca_df, palette="tab10"
18    )
19    plt.show()

```

In [219]:

```
1 plot_pca_clusters(transformed_df, kmeans.labels_)
```



- The clusters above look reasonably well separated.
- This might not always be the case.

Cluster interpretation

- Let's examine the cluster centers and identify types of customers.

In [220]:

```

1 cluster_centers = pd.DataFrame(
2     data=kmeans.cluster_centers_, columns=[transformed_df.columns]
3 )
4 cluster_centers

```

Out[220]:

	CASH_ADVANCE	MINIMUM_PAYMENTS	CASH_ADVANCE_FREQUENCY	PURCHASES_INSTALLMEN
0	-0.155091	0.477421		-0.319096
1	1.688972	0.490910		1.745948
2	-0.182691	-0.119249		-0.101500
3	-0.366373	-0.091844		-0.462599

- Recall that we have applied imputation and scaling on the dataset.
- But we would be able to interpret these clusters better if the centers are in the original scale.
- So let's apply inverse transformations to get the cluster center values in the original scale.

In [221]:

```

1 data = (
2     preprocessor.named_transformers_[ "pipeline" ]
3     .named_steps[ "standardscaler" ]
4     .inverse_transform(cluster_centers[numeric_features])
5 )

```

In [222]:

```

1 org_cluster_centers = pd.DataFrame(data=data, columns=numerical_features)
2 org_cluster_centers = org_cluster_centers.reindex(
3     sorted(org_cluster_centers.columns), axis=1
4 )
5 org_cluster_centers

```

Out[222]:

	BALANCE	BALANCE_FREQUENCY	CASH_ADVANCE	CASH_ADVANCE_FREQUENCY	CASH_ADV.
0	3551.153761	0.986879	653.638891		0.071290
1	4602.462714	0.968415	4520.724309		0.484526
2	1011.751528	0.789871	595.759339		0.114833
3	894.907458	0.934734	210.570626		0.042573

In [223]:

```
1 org_cluster_centers
```

Out[223]:

	BALANCE	BALANCE_FREQUENCY	CASH_ADVANCE	CASH_ADVANCE_FREQUENCY	CASH_ADV.
0	3551.153761	0.986879	653.638891		0.071290
1	4602.462714	0.968415	4520.724309		0.484526
2	1011.751528	0.789871	595.759339		0.114833
3	894.907458	0.934734	210.570626		0.042573

Transactors

- Credit card users who pay off their balance every month with least amount of interest charges.

- They are careful with their money.
- They have lowest balance and cash advance

In [224]: 1 org_cluster_centers

Out[224]:

	BALANCE	BALANCE_FREQUENCY	CASH_ADVANCE	CASH_ADVANCE_FREQUENCY	CASH_ADV
0	3551.153761	0.986879	653.638891		0.071290
1	4602.462714	0.968415	4520.724309		0.484526
2	1011.751528	0.789871	595.759339		0.114833
3	894.907458	0.934734	210.570626		0.042573

Revolvers

- Credit card users who pay off only part of their monthly balance . They use credit card as a loan.
- They have highest balance and cash advance, high cash advance frequency, low purchase frequency, high cash advance transactions, low percentage of full payment
- Their credit limit is also high. (Lucrative group for banks 😞 .)

In [225]: 1 org_cluster_centers

Out[225]:

	BALANCE	BALANCE_FREQUENCY	CASH_ADVANCE	CASH_ADVANCE_FREQUENCY	CASH_ADV
0	3551.153761	0.986879	653.638891		0.071290
1	4602.462714	0.968415	4520.724309		0.484526
2	1011.751528	0.789871	595.759339		0.114833
3	894.907458	0.934734	210.570626		0.042573

VIP/Prime

- Credit card users who have high credit limit.
- They have high one-off purchases frequency, high number of purchase transactions.
- They have high balance but they also have relatively higher percentage of full payment, similar to transactors
- Target for increase credit limit (and increase spending habits)

In [226]: 1 org_cluster_centers

Out[226]:

	BALANCE	BALANCE_FREQUENCY	CASH_ADVANCE	CASH_ADVANCE_FREQUENCY	CASH_ADV
0	3551.153761	0.986879	653.638891		0.071290
1	4602.462714	0.968415	4520.724309		0.484526
2	1011.751528	0.789871	595.759339		0.114833
3	894.907458	0.934734	210.570626		0.042573

Low activity

- Credit card users who have low tenure, low credit limit.
- There is not much activity in the account. It has low balance and not many purchases.

More on interpretation of clusters

- In real life, you'll look through all features in detail before assigning meaning to clusters.
- This is not always easy, especially when you have a large number of features and clusters.
- One way to approach this would be visualizing the distribution of feature values for each cluster as shown below.
- Some domain knowledge would definitely help at this stage.

```
In [227]: 1 # concatenate the cluster labels to our original dataframe
2 creditcard_df_cluster = pd.concat(
3     [creditcard_df, pd.DataFrame({ "cluster": labels})], axis=1
4 )
5 creditcard_df_cluster.head()
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMEN
0	C10001	40.900749	0.818182	95.40		0.00
1	C10002	3202.467416	0.909091	0.00		0.00
2	C10003	2495.148862	1.000000	773.17		773.17
3	C10004	1666.670542	0.636364	1499.00		1499.00
4	C10005	817.714335	1.000000	16.00		16.00

```
In [228]: 1 creditcard_df_cluster[ "cluster" ].unique()
```

```
Out[228]: array([2, 1, 3, 0], dtype=int32)
```

```
In [229]: 1 creditcard_df.columns.shape
```

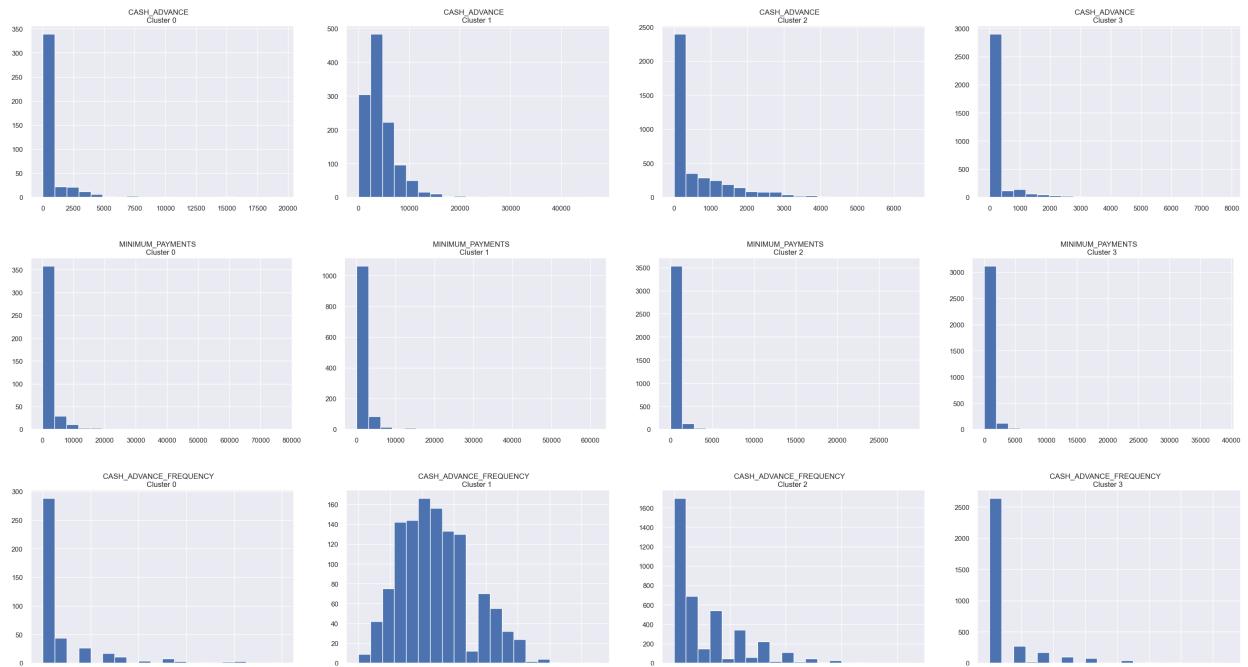
```
Out[229]: (18,)
```

In [230]:

```

1 # Plot the histogram of various clusters
2 for i in transformed_df.columns:
3     plt.figure(figsize=(35, 5))
4     for j in range(4):
5         plt.subplot(1, 4, j + 1)
6         cluster = creditcard_df_cluster[creditcard_df_cluster["cluster"]
7         cluster[i].hist(bins=20)
8         plt.title("{}\nCluster {}".format(i, j))
9
10    plt.show()

```



Practice exercise for you

- Try out different values for `n_clusters` in `KMeans` and examine the clusters.
- If you are feeling adventurous, you may try customer segmentation on [All Lending Club loan data](#) (<https://www.kaggle.com/wordsforthewise/lending-club>).

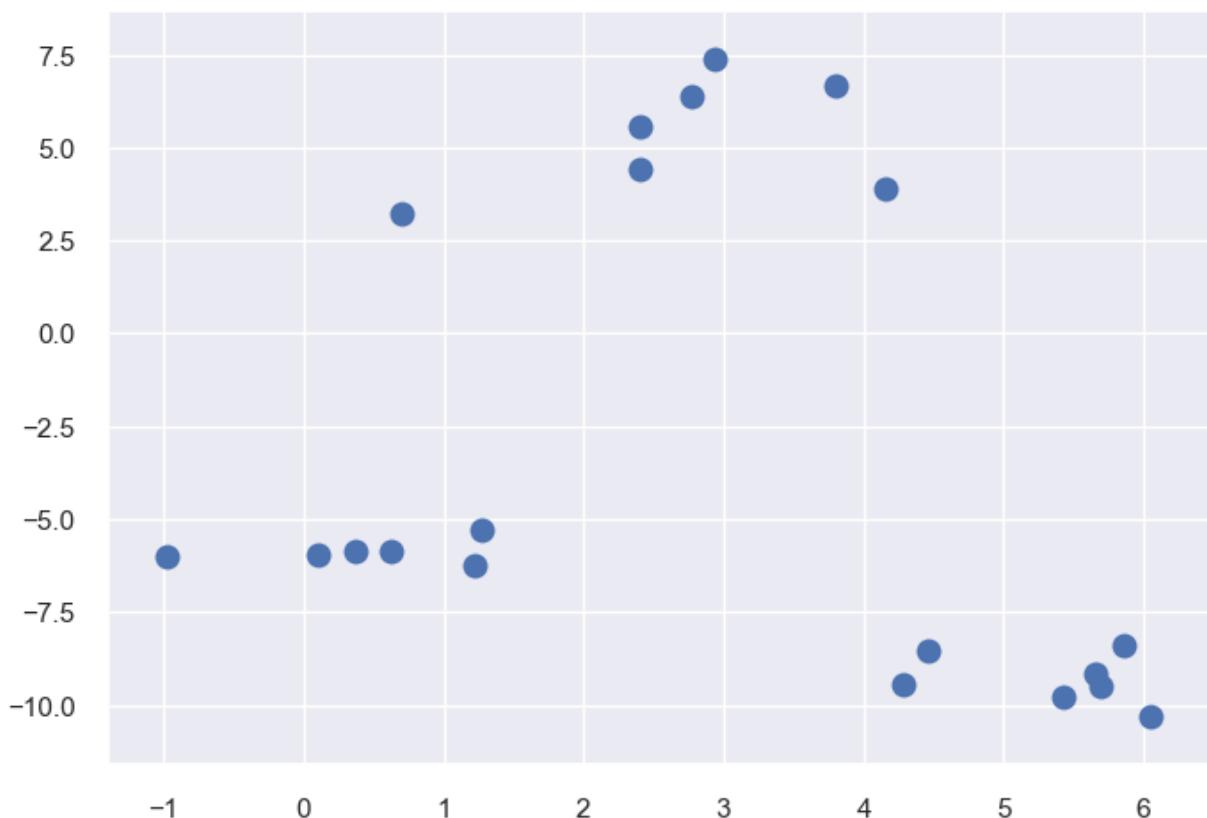
Final comments and summary

A comment on initialization

- The initialization of K-Means is stochastic, can this affect the results?
 - Yes! Big time.

In [231]:

```
1 X, y = make_blobs(n_samples=20, centers=3, n_features=2, random_state=1)
2 mglearn.discrete_scatter(X[:, 0], X[:, 1], markers="o");
```



In [232]:

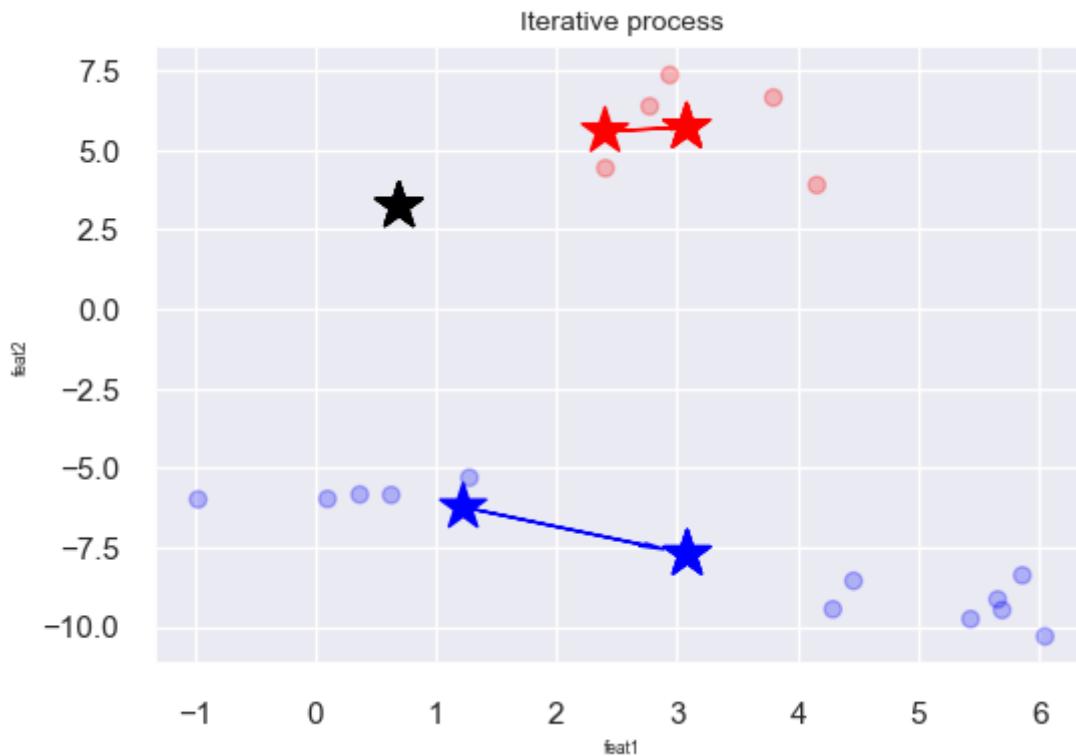
```
1 k = 3
2 n_examples = X.shape[0]
3 toy_df = pd.DataFrame(X, columns=["feat1", "feat2"])
```

Example: Bad initialization

In [233]:

```
1 np.random.seed(seed=10)
2 centroids_idx = np.random.choice(range(0, n_examples), size=k)
3 centroids = X[centroids_idx]
```

```
In [234]: 1 plot_iterative(toy_df, 6, 4, centroids)
```

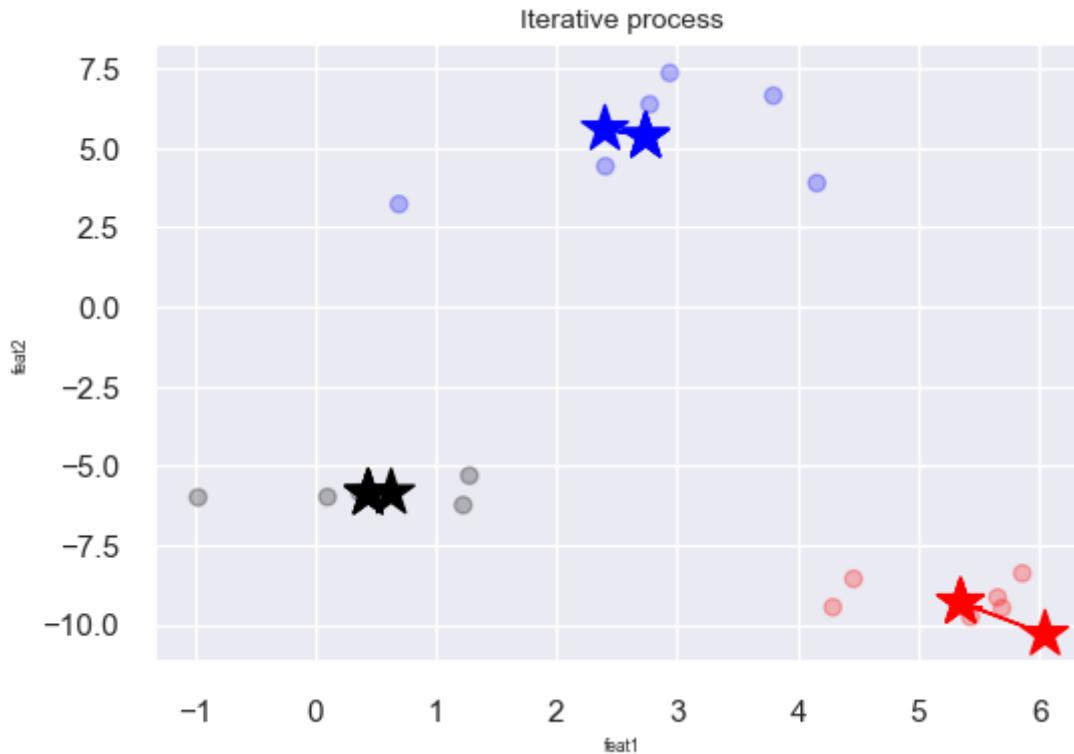


Example: Better initialization

The following initialization seems much better.

```
In [235]: 1 np.random.seed(seed=2)
2 centroids_idx = np.random.choice(range(0, n_examples), size=k)
3 centroids = X[centroids_idx]
```

```
In [236]: 1 plot_iterative(toy_df, 6, 4, centroids)
```



What can we do about it?

- One strategy is to run the algorithm several times.
 - Check out `n_init` parameter of `sklearn's KMeans` (<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>).
- Is it possible to pick k in a smart way?
 - Yes! We can use the so-called "K-Means++".
 - Intuitively, it picks the initial centroids which are far away from each other.
 - In other words, K-Means++ gives more chance to select points that are far away from centroids already picked.
 - By default `sklearn` uses this strategy for initialization.

Important points to remember

- Clustering is a common unsupervised approach to identify underlying structure in data and grouping points based on similarity.
- K-Means is a popular clustering algorithm.

Important points to remember

K-Means

- It requires us to specify the number of clusters in advance.

- Each example is assigned to one (and only one) cluster.
- The labels provided by the algorithm have no actual meaning.
- The centroids live in the same space as of the dataset but they are **not** actual data points, but instead are average points.
- It always converges. Convergence is dependent upon the initial centers and it may converge to a sub-optimal solution.

Important points to remember

- Two ways to provide insight into how many clusters are reasonable for the give problem are: **the Elbow method** and **the Silhouette method**.
- Some applications of K-Means clustering include data exploration, feature engineering, customer segmentation, and document clustering.
- It takes fair amount of manual effort and domain knowledge to interpret clusters returned by K-Means.

Resources

- ["Spaghetti Sauce" talk by Malcom Gladwell](https://www.ted.com/talks/malcolm_gladwell_on_spaghetti_sauce?language=en)
- [Visualizing_k-means-clustering_](https://www.naftaliharris.com/blog/visualizing-k-means-clustering/)
- [Visualizing K-Means algorithm with D3.js](http://tech.nitoyon.com/en/blog/2013/11/07/k-means/)
- [Clustering with Scikit with GIFs](https://dashee87.github.io/data%20science/general/Clustering-with-Scikit-with-GIFs/)
- [sklearn clustering documentation](https://scikit-learn.org/stable/modules/clustering.html)

CPSC 330

Applied Machine Learning

Lecture 15: DBSCAN and Hierarchical Clustering

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

In [1]:

```
1 import os
2 import random
3 import sys
4
5 import numpy as np
6 import pandas as pd
7
8 sys.path.append("../code/.")
9 from plotting_functions import *
10 from plotting_functions_unsup import *
11 import matplotlib.pyplot as plt
12 import mglearn
13 import seaborn as sns
14 from ipywidgets import interactive
15 from plotting_functions import *
16 from plotting_functions_unsup import *
17 from scipy.cluster.hierarchy import dendrogram, fcluster, linkage
18 from sklearn import cluster, datasets, metrics
19 from sklearn.cluster import DBSCAN, AgglomerativeClustering, KMeans
20 from sklearn.datasets import make_blobs, make_moons
21 from sklearn.decomposition import PCA
22 from sklearn.metrics.pairwise import euclidean_distances
23 from sklearn.preprocessing import StandardScaler
24 from yellowbrick.cluster import SilhouetteVisualizer
25
26 plt.rcParams["font.size"] = 16
27 %matplotlib inline
28 pd.set_option("display.max_colwidth", 0)
```

Announcements

- Homework 6 due tomorrow (Mar 15).
- Homework 7 will be released on Thursday (March 16).
- Final exam announced: April 20, at 19:00 PM (in CIRS 1250 or BIOL 1000, stay tuned for how we will use rooms).

Learning outcomes

From this lecture, students are expected to be able to:

- Identify limitations of K-Means.

- Broadly explain how DBSCAN works.
- Apply DBSCAN using `sklearn`.
- Explain the effect of epsilon and minimum samples hyperparameters in DBSCAN.
- Explain the difference between core points, border points, and noise points in the context of DBSCAN.
- Identify DBSCAN limitations.
- Explain the idea of hierarchical clustering.
- Visualize dendrograms using `scipy.cluster.hierarchy.dendrogram`.
- Explain the advantages and disadvantages of different clustering methods.
- Apply clustering algorithms on image datasets and interpret clusters.
- Recognize the impact of distance measure and representation in clustering methods.

Recap and motivation

K-Means recap

- We discussed K-Means clustering in the previous lecture.
- Each cluster is represented by a center.
- Given a new point, you can assign it to a cluster by computing the distances to all cluster centers and picking the cluster with the smallest distance.
- It's a popular algorithm because
 - It's easy to understand and implement.
 - Runs relatively quickly and scales well to large datasets.
 - `sklearn` has a more scalable variant called [MiniBatchKMeans](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html) (<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html>) which can handle very large datasets.

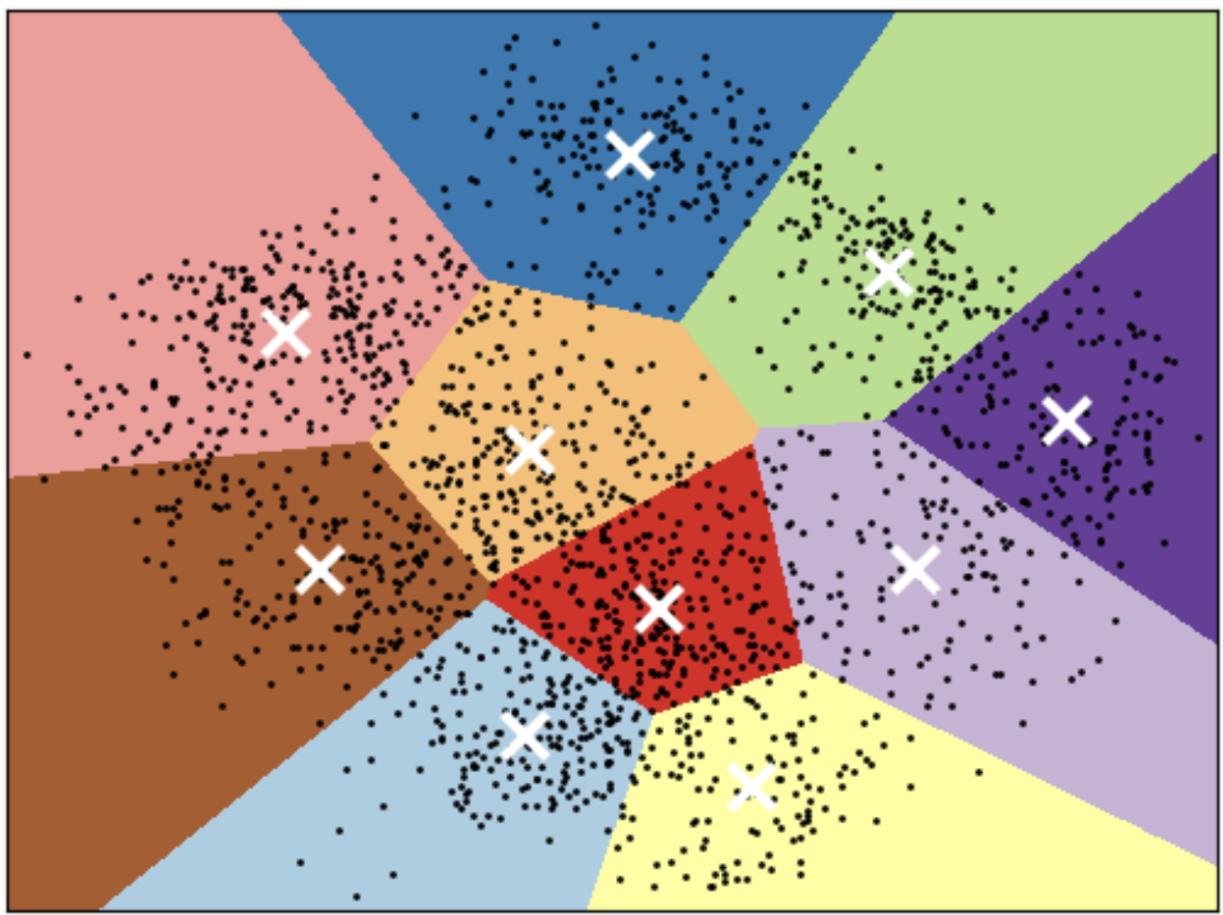
K-Means limitations

- Relies on random initialization and so the outcome may change depending upon this initialization.
- K-Means clustering requires to specify the number of clusters in advance.
- Very often you do not know the centers in advance. The elbow method or the silhouette method to find the optimal number of clusters are not always easy to interpret.
- Each point has to have a cluster assignment.

K-Means limitations: Shape of K-Means clusters

- K-Means partitions the space based on the closest mean.

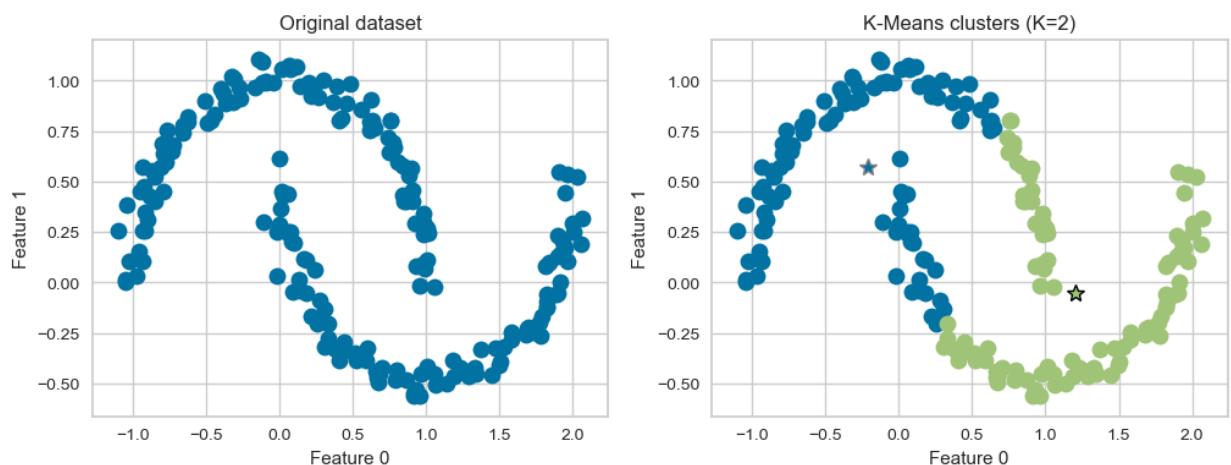
- Each cluster is defined solely by its center and so it can only capture relatively simple shapes.
- So the boundaries between clusters are linear; It fails to identify clusters with complex shapes.
[Source \(\[https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html\]\(https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html\)\).](https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html)



K-Means: failure case 1

- K-Means performs poorly if the clusters have more complex shapes (e.g., two moons data below).

```
In [2]: 1 X, y = make_moons(n_samples=200, noise=0.05, random_state=42)
2 plot_X_k_means(X, k=2)
```



K-Means: failure case 2

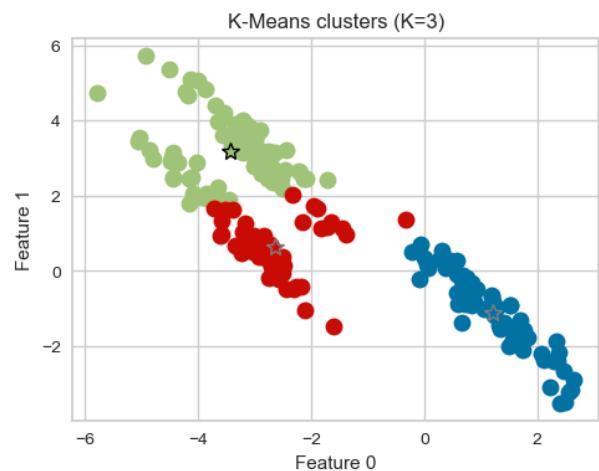
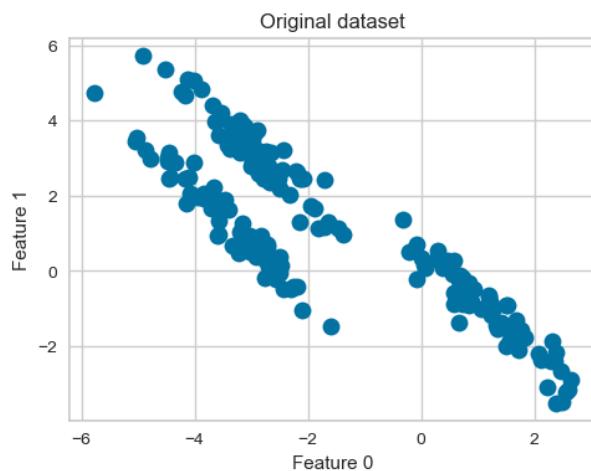
- It assumes that all directions are equally important for each cluster and fails to identify non-spherical clusters.

In [3]:

```

1 # generate some random cluster data
2 X, y = make_blobs(random_state=170, n_samples=200)
3 rng = np.random.RandomState(74)
4 transformation = rng.normal(size=(2, 2))
5 X = np.dot(X, transformation)
6 plot_X_k_means(X, 3)

```



K-Means: failure case 3

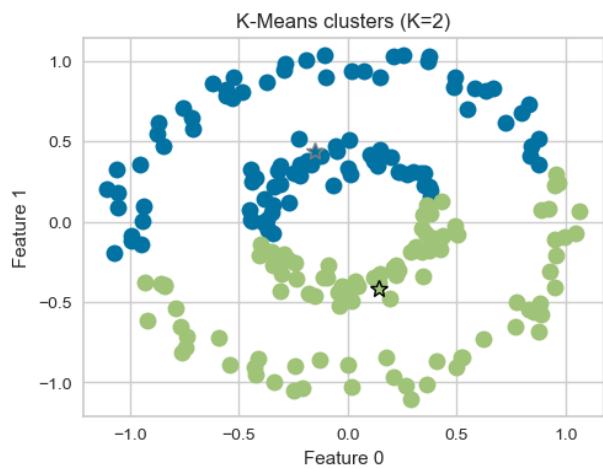
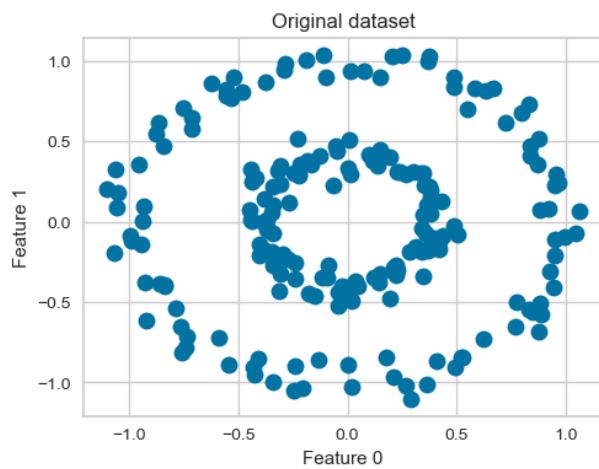
- Again, K-Means is unable to capture complex cluster shapes.

In [4]:

```

1 X = datasets.make_circles(n_samples=200, noise=0.06, factor=0.4)[0]
2 plot_X_k_means(X, 2)

```



- Can we do better than this?

DBSCAN

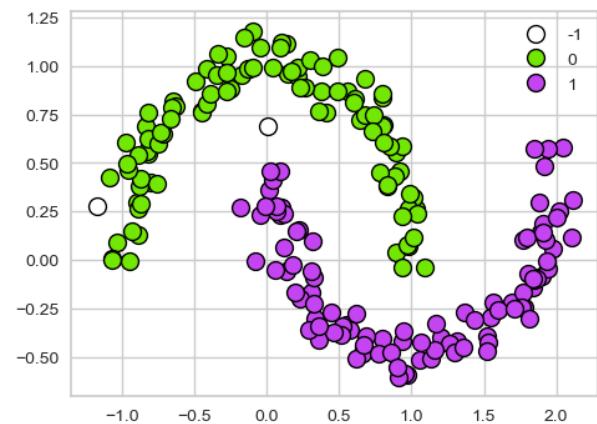
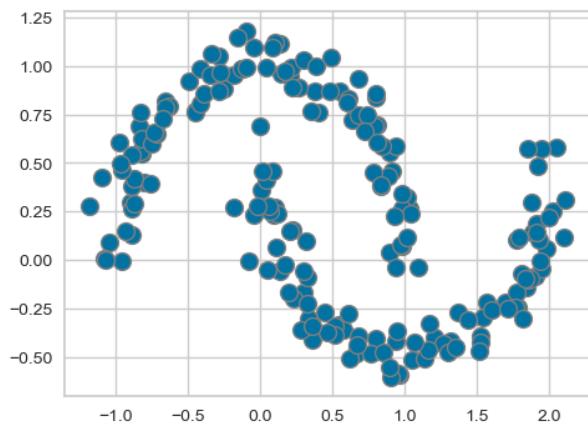
- Density-Based Spatial Clustering of Applications with Noise

DBSCAN introduction

- DBSCAN is a density-based clustering algorithm.
- Intuitively, it's based on the idea that clusters form dense regions in the data and so it works by identifying "crowded" regions in the feature space.
- It can address some of the limitations of K-Means we saw above.
 - It does not require the user to specify the number of clusters in advance.
 - It can identify points that are not part of any clusters.
 - It can capture clusters of complex shapes.

Let's try `sklearn`'s DBSCAN.

```
In [5]: 1 x, y = make_moons(n_samples=200, noise=0.08, random_state=42)
2 dbSCAN = DBSCAN(eps=0.2)
3 dbSCAN.fit(X)
4 plot_X_dbSCAN(X, dbSCAN)
```



```
In [6]: 1 dbSCAN.labels_
```

```
Out[6]: array([[ 0,  0,  0,  1,  1,  1,  0,  1,  1,  0,  0,  1,  1,  1,  0,  1,  1,
  0,  1,  0,  0,  1,  0,  1,  0,  0,  0,  0,  0,  1,  1,  1,  0,  0,  0,
  0,  1,  1,  1,  1,  1,  0,  0,  1,  0,  0,  1,  1,  1,  0,  0,  1,  0,
  1,  1,  0,  1,  1,  0,  1,  0,  1,  0,  1,  0,  0,  1,  1,  1,  0,  0,  0,
  1,  1,  0,  1,  1,  0,  1,  0,  1,  1,  0,  0,  1,  1,  1,  0,  0,  0,  0,
  1,  0,  1,  1,  1,  0,  1,  0,  1,  0,  1,  0,  0,  1,  1,  0,  0,  0,  0,
  1,  0,  1,  0,  1,  1,  0,  1,  1,  0,  1,  1,  0,  1,  0,  0,  0,  0,  1,
  1,  0,  1,  0,  1,  1,  0,  1,  1,  0,  1,  1,  0,  1,  0,  0,  0,  0,  1,
  0,  1,  1,  1,  1,  0,  0,  1,  1,  1,  0,  1,  1,  1,  0,  1,  1,  0,  0,
  1,  1,  0,  1,  1,  0,  0,  1,  1,  1,  0,  1,  1,  1,  0,  1,  1,  0,  0,
  1,  0,  0,  1,  1,  1,  0,  0,  1,  1,  1,  0,  1,  1,  1,  0,  0,  0,  0,
  0,  1,  0,  1,  1,  1,  0,  0,  1,  1,  1,  0,  1,  1,  1,  0,  0,  0,  0,
  0,  0,  1,  0,  1,  1,  1,  0,  0,  1,  1,  1,  0,  1,  1,  1,  0,  0,  0,
  0,  0,  0,  1,  0,  1,  1,  1,  0,  0,  1,  1,  1,  0,  1,  1,  1,  0,  0,
  0,  0,  0,  0,  1,  1,  1,  1,  0,  0,  1,  1,  1,  0,  1,  1,  1,  0,  0,
  0,  1,  0,  0,  1,  1,  0,  1,  1,  0,  1,  0,  1,  1,  0,  -1,  1,  1]])
```

- DBSCAN is able to capture half moons shape
 - We don't have to specify the number of clusters.
 - That said, it has two other non-trivial hyperparameters to tune.
 - There are two examples which have not been assigned any label (noise examples).

One more example of DBSCAN clusters capturing complex cluster shapes.

```
In [71]: 1 X = datasets.make_circles(n_samples=200, noise=0.06, factor=0.4)[0]
          2 dbSCAN = DBSCAN(eps=0.2, min_samples=3)
          3 dbSCAN.fit(X)
          4 plot_X_dbSCAN(X, dbSCAN)
```

How does it work?

- Iterative algorithm.
 - Based on the idea that clusters form dense regions in the data.

Source (<https://dashee87.github.io/data%20science/general/Clustering-with-Scikit-with-GIFs/>)

Two main hyperparameters

- `eps` : determines what it means for points to be "close"

- `min_samples` : determines the number of **neighboring points** we require to consider in

Effect of `eps` hyperparameter

- `eps` : determines what it means for points to be "close"

```
In [8]: 1 x, y = make_blobs(random_state=0, n_samples=11)
```

```
In [72]: 1 # interactive(lambda eps=1: plot_dbscan_with_labels(X, eps), eps=(1, 12)
2 plot_dbscan_with_labels(X, eps=1)
```

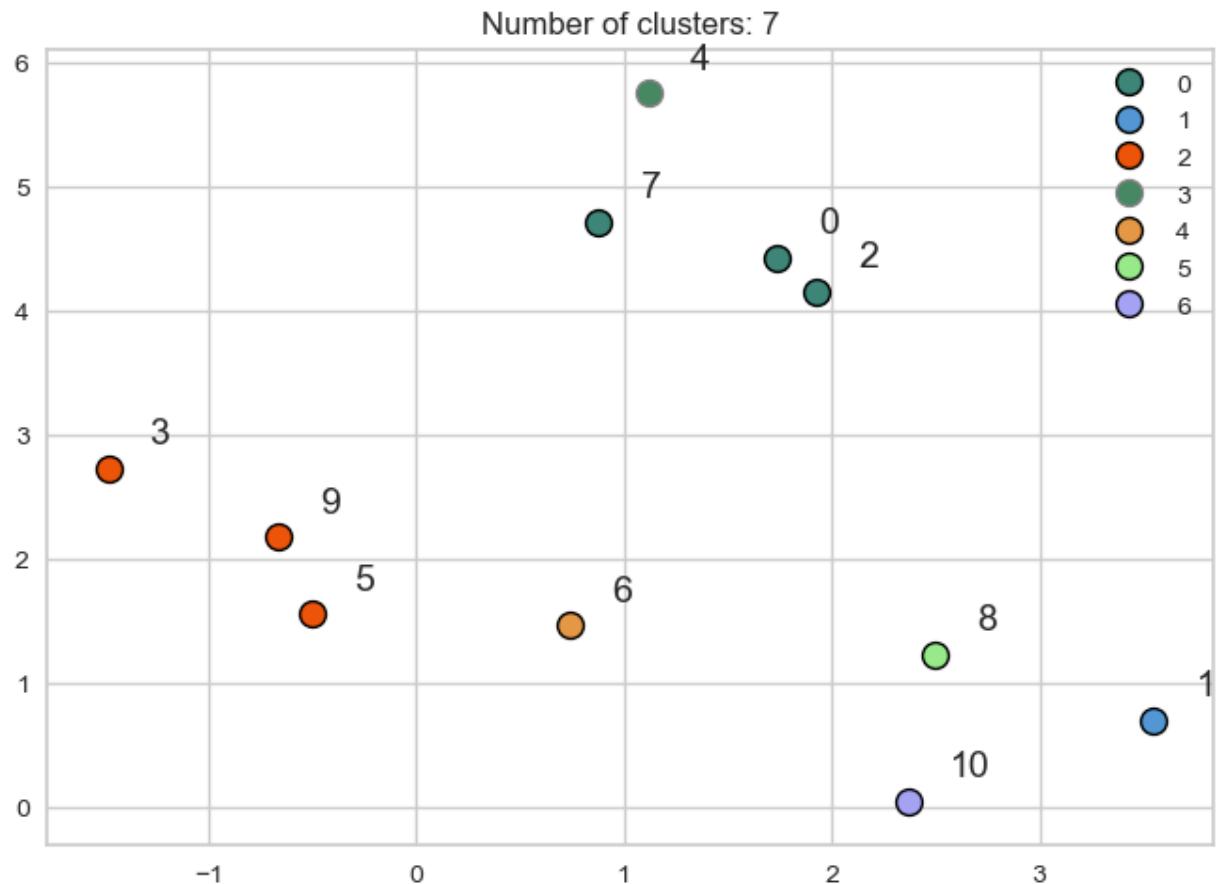
```
In [10]: 1 pd.DataFrame(euclidean_distances(X, X))
```

Out[10]:

	0	1	2	3	4	5	6	7	8	
0	0.000000	4.149512	0.332095	3.637398	1.468503	3.640916	3.124431	0.911272	3.283718	3.2894
1	4.149512	0.000000	3.821671	5.427714	5.617833	4.137181	2.910477	4.830950	1.180379	4.4653
2	0.332095	3.821671	0.000000	3.692209	1.796596	3.555578	2.937847	1.193839	2.976718	3.2572
3	3.637398	5.427714	3.692209	0.000000	3.991555	1.535625	2.559670	3.078516	4.253314	0.9890
4	1.468503	5.617833	1.796596	3.991555	0.000000	4.507219	4.311101	1.072565	4.732056	4.0014
5	3.640916	4.137181	3.555578	1.535625	4.507219	0.000000	1.243674	3.447353	3.013883	0.6458
6	3.124431	2.910477	2.937847	2.559670	4.311101	1.243674	0.000000	3.253476	1.771561	1.5754
7	0.911272	4.830950	1.193839	3.078516	1.072565	3.447353	3.253476	0.000000	3.843925	2.9666
8	3.283718	1.180379	2.976718	4.253314	4.732056	3.013883	1.771561	3.843925	0.000000	3.2996
9	3.289419	4.465344	3.257210	0.989059	4.001437	0.645802	1.575484	2.966927	3.299630	0.0000
10	4.427098	1.347556	4.132565	4.694590	5.849187	3.238439	2.158345	4.904326	1.194950	3.7056

Effect of min_samples hyperparameter

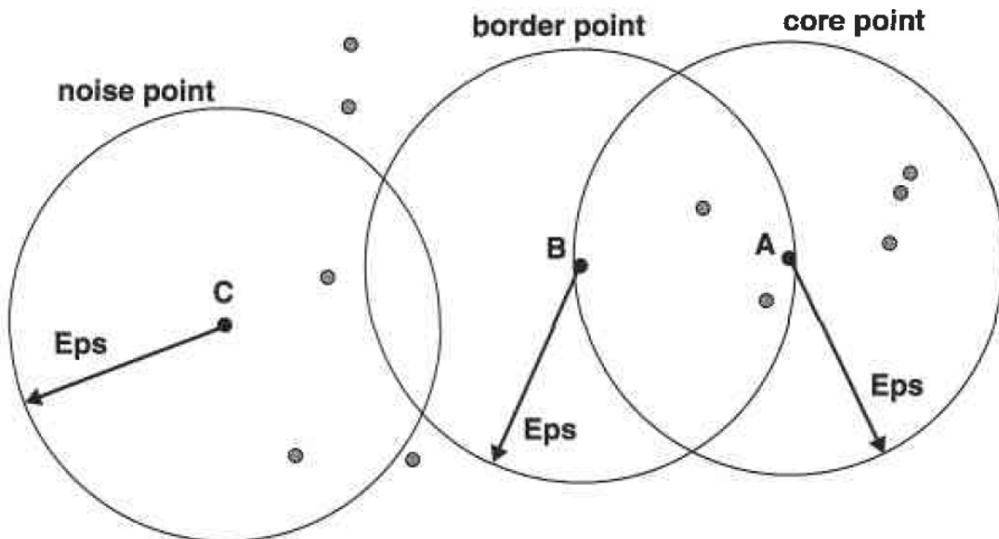
```
In [11]: 1 # interactive(lambda min_samples=1: plot_dbscan_with_labels(X, eps=1.0,
2 plot_dbscan_with_labels(X, eps=1.0, min_samples=1)
```



More details on DBSCAN

There are three kinds of points.

- **Core points** are the points that have at least `min_samples` points in the neighborhood.
- **Border points** are the points with fewer than `min_samples` points in the neighborhood, but are connected to a core point.
- **Noise points** are the points which do not belong to any cluster. In other words, the points which have less than `min_samples` point within distance `eps` of the starting point are noise points.



DBSCAN algorithm

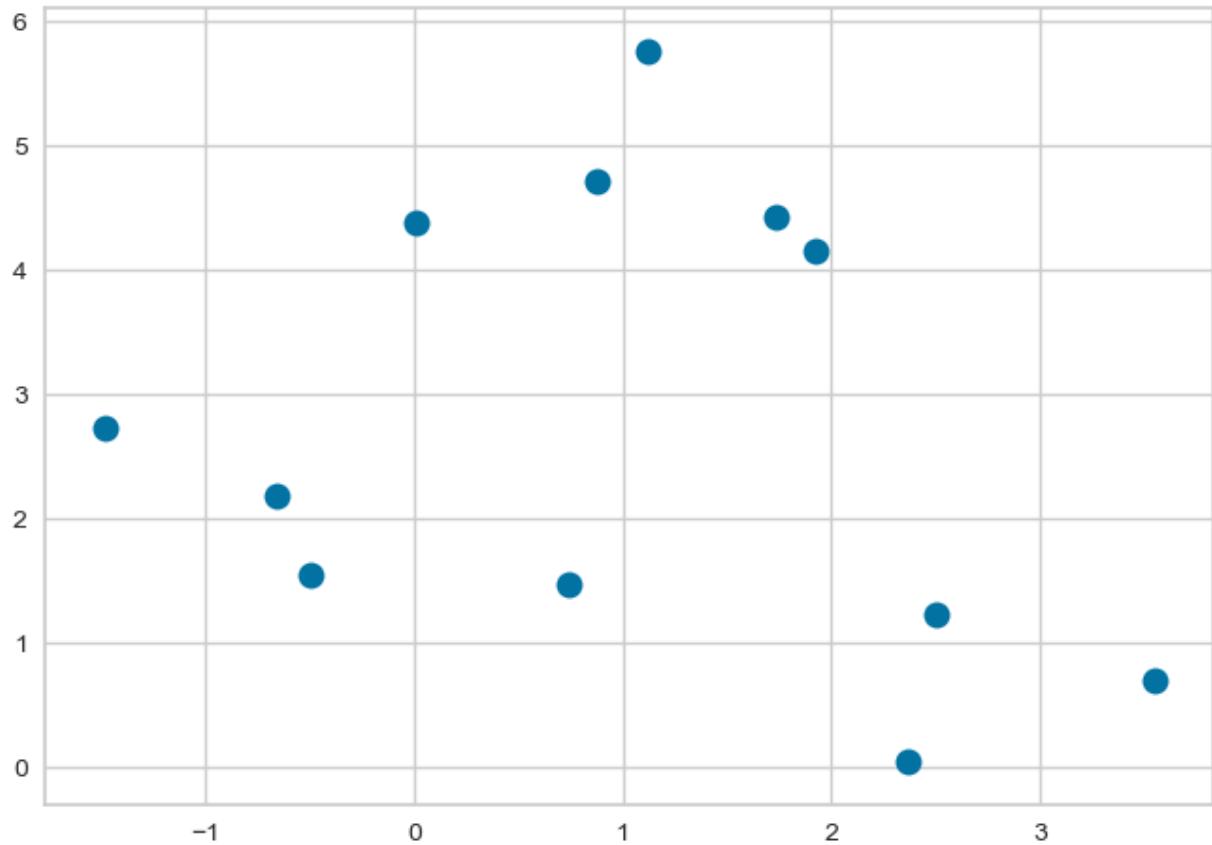
- Pick a point p at random.
- Check whether p is a "core" point or not. You can check this by looking at the number of neighbours within epsilon distance if they have at least `min_samples` points in the neighbourhood
- If p is a core point, give it a colour (label).
- Spread the colour of p to all of its neighbours.
- Check if any of the neighbours that received the colour is a core point, if yes, spread the colour to its neighbors as well.
- Once there are no more core points left to spread the colour, pick a new unlabeled point p and repeat the process.

K-Means vs. DBSCAN

- In DBSCAN, you do not have to specify the number of clusters!
 - Instead, you have to tune `eps` and `min_samples`.
- Unlike K-Means, DBSCAN doesn't have to assign all points to clusters.
 - The label is `-1` if a point is unassigned.
- Unlike K-Means, there is no `predict` method.
 - DBSCAN only really clusters the points you have, not "new" or "test" points.

Illustration of hyperparameters `eps` and `min_samples`

```
In [12]: 1 X, y = make_blobs(random_state=0, n_samples=12)
2 mglearn.discrete_scatter(X[:, 0], X[:, 1]);
```



```
In [13]: 1 dbscan = DBSCAN()
2 clusters = dbscan.fit_predict(X)
3 print("Cluster memberships:{}".format(clusters))
```

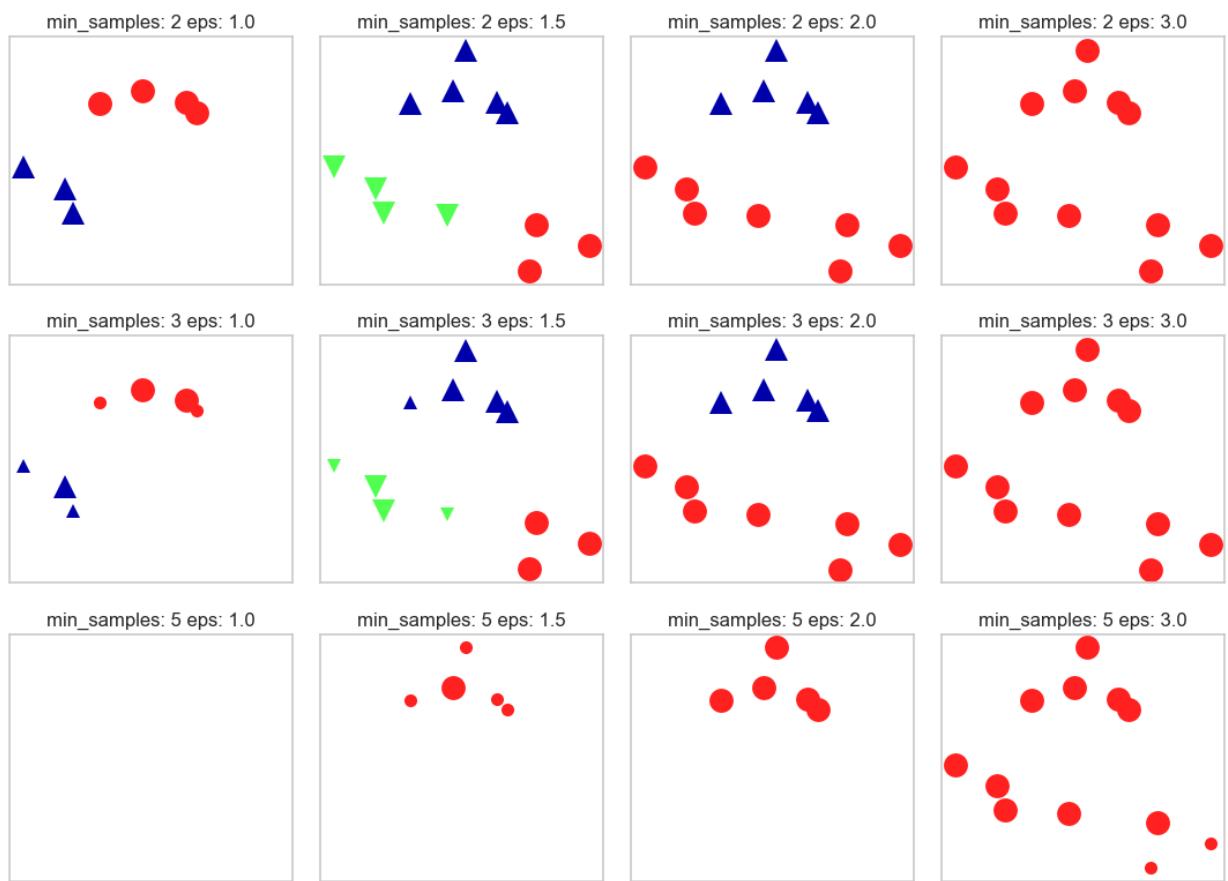
Cluster memberships:[-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]

- Default values for hyperparameters don't work well on toy datasets.
- All points have been marked as noise with the default values for `eps` and `min_samples`
- Let's examine the effect of changing these hyperparameters.
 - noise points: shown in white
 - core points: bigger
 - border points: smaller

In [14]:

```
1 mglearn.plots.plot_dbSCAN()
```

```
min_samples: 2 eps: 1.000000 cluster: [-1  0  0 -1  0 -1  1  1  0  1 -1
-1]
min_samples: 2 eps: 1.500000 cluster: [0  1  1  1  1  0  2  2  1  2  2  0]
min_samples: 2 eps: 2.000000 cluster: [0  1  1  1  1  0  0  0  1  0  0  0]
min_samples: 2 eps: 3.000000 cluster: [0  0  0  0  0  0  0  0  0  0  0  0]
min_samples: 3 eps: 1.000000 cluster: [-1  0  0 -1  0 -1  1  1  0  1 -1
-1]
min_samples: 3 eps: 1.500000 cluster: [0  1  1  1  1  0  2  2  1  2  2  0]
min_samples: 3 eps: 2.000000 cluster: [0  1  1  1  1  0  0  0  1  0  0  0]
min_samples: 3 eps: 3.000000 cluster: [0  0  0  0  0  0  0  0  0  0  0  0]
min_samples: 5 eps: 1.000000 cluster: [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1]
min_samples: 5 eps: 1.500000 cluster: [-1  0  0  0  0 -1 -1 -1  0 -1 -1 -1
-1]
min_samples: 5 eps: 2.000000 cluster: [-1  0  0  0  0 -1 -1 -1  0 -1 -1 -1
-1]
min_samples: 5 eps: 3.000000 cluster: [0  0  0  0  0  0  0  0  0  0  0  0]
```



Observations

- Increasing `eps` (\uparrow) (left to right in the plot above) means more points will be included in a cluster.
 - `eps = 1.0` either creates more clusters or more noise points, whereas `eps=3.0` puts all points in one cluster with no noise points.
- Increasing `min_samples` (\uparrow) (top to bottom in the plot above) means points in less dense regions will either be labeled as their own cluster or noise.

- `min_samples=2`, for instance, has none or only a few noise points whereas `min_samples=5` has several noise points.
- Here `min_samples = 2.0` or `3.0` and `eps = 1.5` is giving us the best results.
- In general, it's not trivial to tune these hyperparameters.

Question for you

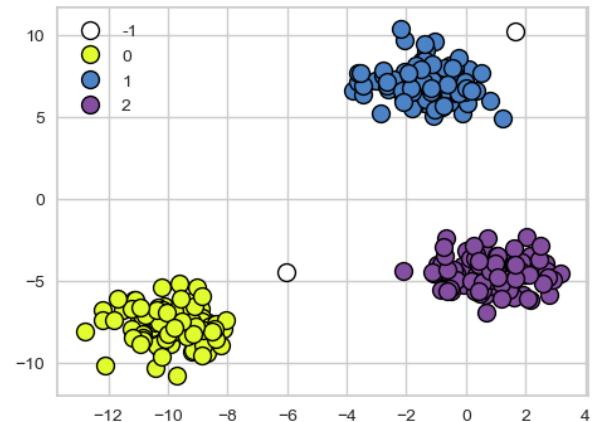
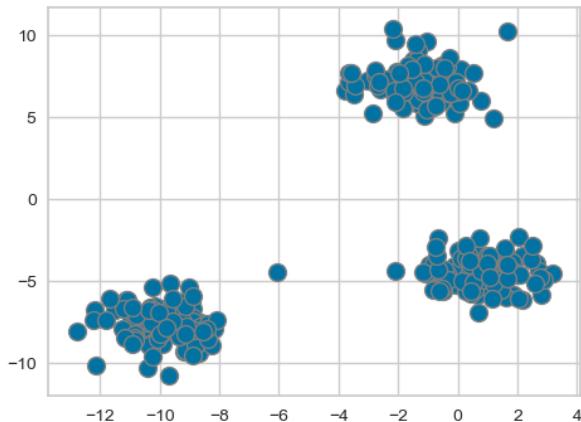
- Does the order that you pick the points matter in DBSCAN?

No. Any of the cluster's core points is able to fully identify the cluster, with no randomness involved. The only possible conflict you might get is that if two clusters have the same border point. In this case the assignment will be implementation dependent, but usually the border point will be assigned to the first cluster that "finds" it.

Evaluating DBSCAN clusters

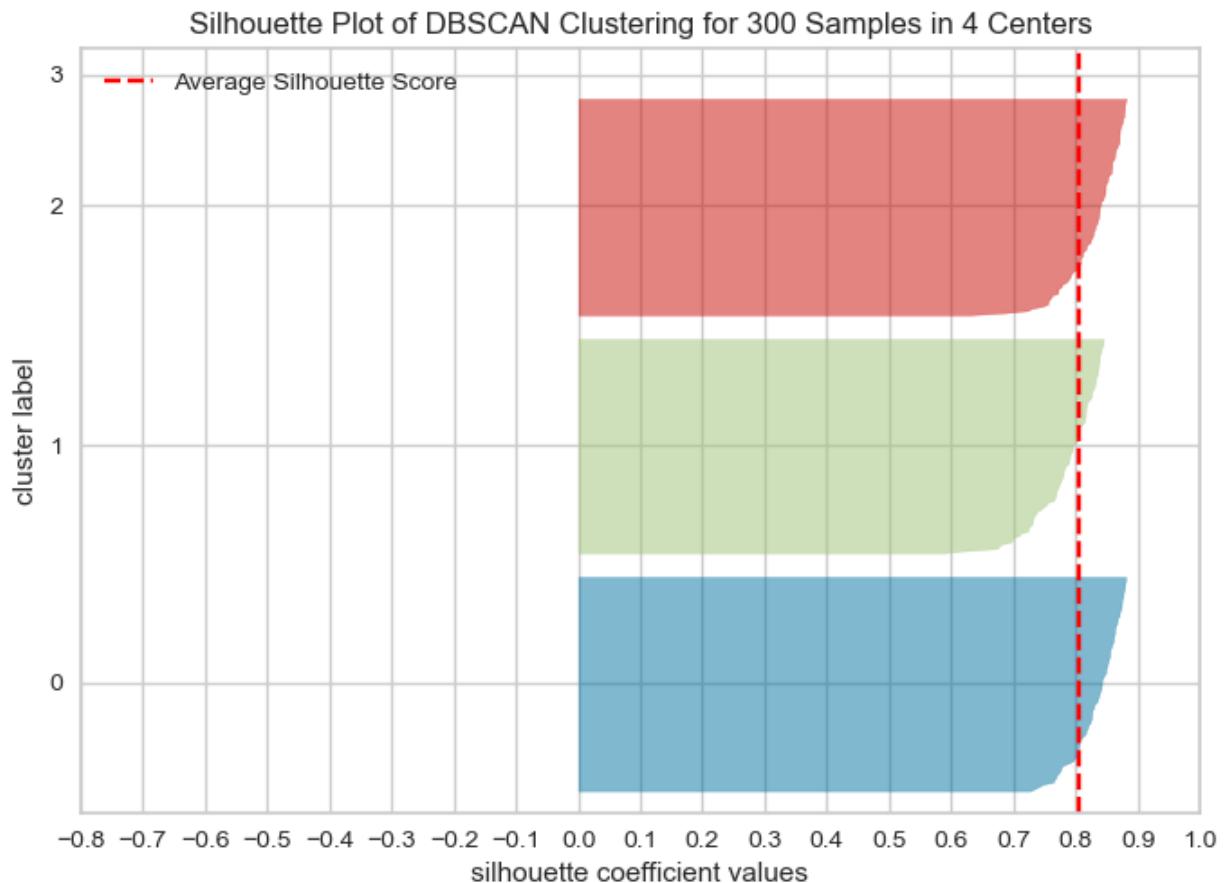
- We cannot use the elbow method to examine the goodness of clusters created with DBSCAN.
- But we can use the silhouette method because it's not dependent on the idea of cluster centers.

```
In [15]: 1 X, y = make_blobs(random_state=100, centers=3, n_samples=300)
2 dbSCAN = DBSCAN(eps=2, min_samples=5)
3 dbSCAN.fit(X)
4 plot_X_dbSCAN(X, dbSCAN)
```



```
In [16]: 1 # Yellowbrick is designed to work with K-Means and not with DBSCAN.
2 # So it needs the number of clusters stored in n_clusters
3 # It also needs `predict` method to be implemented.
4 # So I'm implementing it here so that we can use Yellowbrick to show Si
5 n_clusters = len(set(dbSCAN.labels_))
6 dbSCAN.n_clusters = n_clusters
7 dbSCAN.predict = lambda x: dbSCAN.labels_
```

```
In [17]: 1 visualizer = SilhouetteVisualizer(dbSCAN, colors="yellowbrick")
2 visualizer.fit(X) # Fit the data to the visualizer
3 visualizer.show();
```



Summary: Pros and cons

- Pros
 - Can learn arbitrary cluster shapes
 - Can detect outliers
- Cons
 - Cannot predict on new examples.
 - Needs tuning of two non-obvious hyperparameters

There is an improved version of DBSCAN called [HDBSCAN \(hierarchical DBSCAN\)](#). (<https://github.com/scikit-learn-contrib/hdbscan>).

DBSCAN: failure cases

- DBSCAN is able to capture complex clusters. But this doesn't mean that DBSCAN always works better. It has its own problems!
- DBSCAN doesn't do well when we have clusters with different densities.
 - You can play with the hyperparameters but it's not likely to help much.

DBSCAN: failure cases

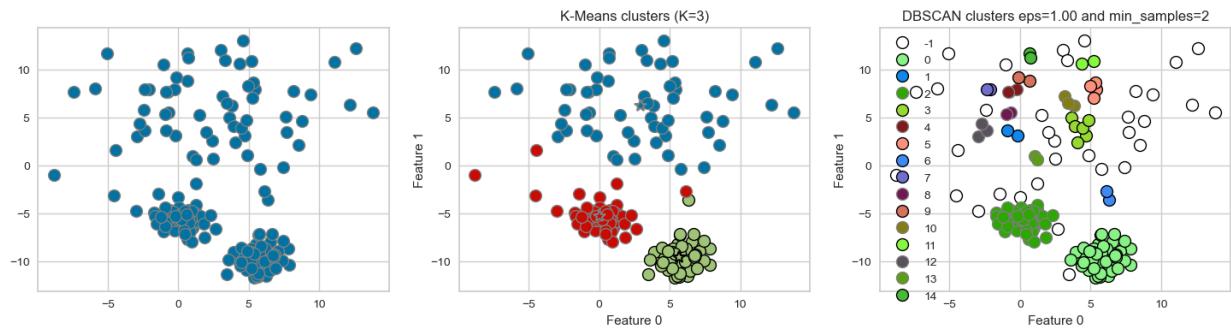
- Let's consider this dataset with three clusters of varying densities.
- K-Means performs better compared to DBSCAN. But it has the benefit of knowing the value of K in advance.

In [18]:

```

1 X_varied, y_varied = make_blobs(
2     n_samples=200, cluster_std=[1.0, 5.0, 1.0], random_state=10
3 )
4 plot_k_means_dbSCAN_comparison(X_varied)

```



Hierarchical clustering

Motivation

- Deciding how many clusters we want is a hard problem.
- Often, it's useful to get a complete picture of similarity between points in our data before picking the number of clusters.
- Hierarchical clustering is helpful in these scenarios.

Main idea

1. Start with each point in its own cluster.
2. Greedily merge most similar *clusters*.
3. Repeat Step 2 until you obtain only one cluster ($n-1$ times).

Visualizing hierarchical clustering

- Hierarchical clustering can be visualized using a tool called **a dendrogram**.
- Unfortunately, `sklearn` cannot do it so we will use the package `scipy.cluster.hierarchy` for hierarchical clustering.

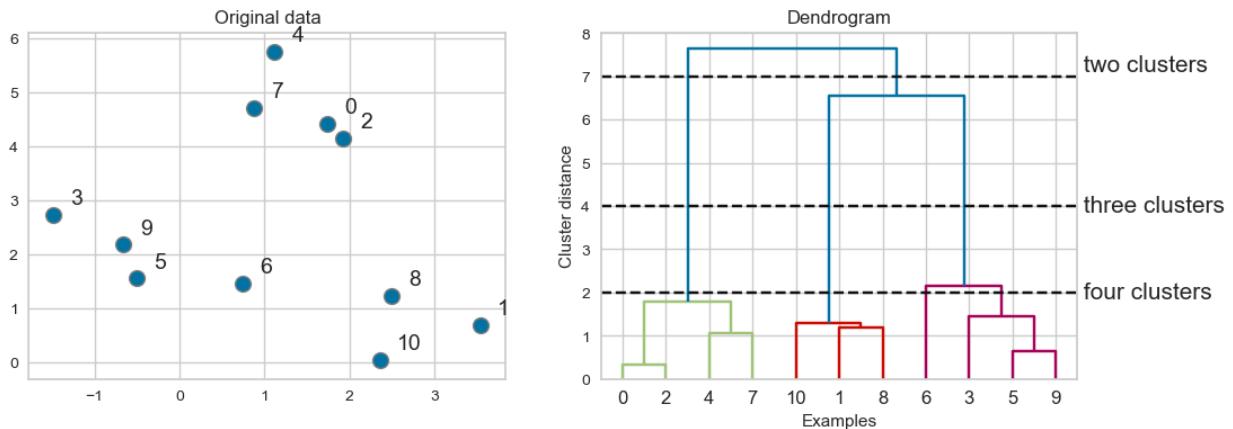
Hierarchical clustering input and output

In [19]:

```
1 import scipy
2 X, y = make_blobs(random_state=0, n_samples=11)
3 linkage_array = scipy.cluster.hierarchy.ward(X)
```

In [20]:

```
1 plot_X_dendrogram(X, linkage_array, label_n_clusters=True)
```



- Every point goes through the journey of being on its own (its own cluster) and getting merged with some other bigger clusters.
- The intermediate steps in the process provide us clustering with different number of clusters.

Dendrogram

- Dendrogram is a tree-like plot.
- On the x-axis we have data points.
- On the y-axis we have distances between clusters.
- We start with data points as leaves of the tree.
- New parent node is created for every two clusters that are joined.
- The length of each branch shows how far the merged clusters go.

- In the dendrogram above going from three clusters to two clusters means merging far apart points because the branches between three cluster to two clusters are long.

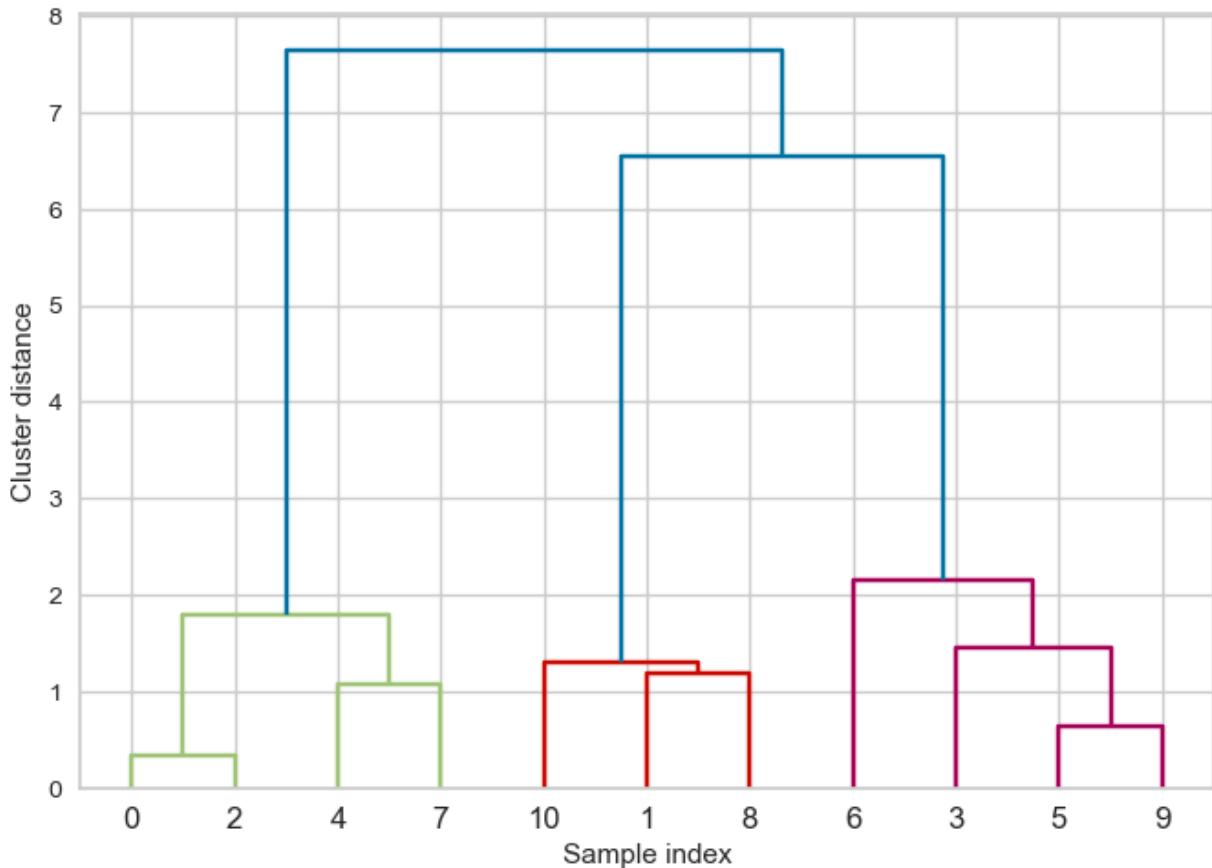
How to plot a dendrogram?

In [21]:

```

1 from scipy.cluster.hierarchy import dendrogram
2
3 ax = plt.gca()
4 dendrogram(linkage_array, ax=ax)
5 plt.xlabel("Sample index")
6 plt.ylabel("Cluster distance");

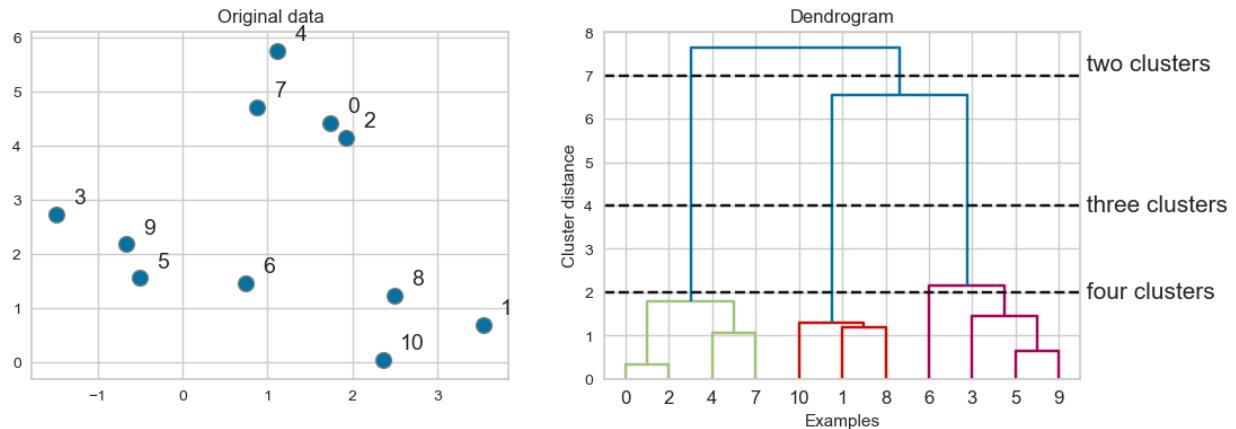
```



What do we mean by distance between clusters?

- We know how to measure distance between points (e.g., using Euclidean distance).
- How do we measure distances between clusters?
- The **linkage criteria** determines how to find similarity between clusters:
- Some example linkage criteria are:
 - single linkage (minimum distance)
 - average linkage (average distance)
 - complete (or maximum) linkage (maximum distance)
 - ward linkage

In [22]: 1 plot_X_dendrogram(X, linkage_array, label_n_clusters=True)



single linkage

- Merges two clusters that have the smallest minimum distance between all their points.
- Let's use `scipy.cluster.hierarchy's` `single` to get linkage information.
- This method gives us matrix `z` with the merging information.

In [23]: 1 from scipy.cluster.hierarchy import (

```

2     average,
3     complete,
4     dendrogram,
5     fcluster,
6     single,
7     ward,
8 )
9
10 Z = single(X)
11 columns = ["c1", "c2", "distance(c1, c2)", "# observations"]
```

In [24]: 1 pd.DataFrame(Z, columns=columns).head()

Out[24]:

	c1	c2	distance(c1, c2)	# observations
0	0.0	2.0	0.332095	2.0
1	5.0	9.0	0.645802	2.0
2	7.0	11.0	0.911272	3.0
3	3.0	12.0	0.989059	3.0
4	4.0	13.0	1.072565	4.0

- The linkage returns a matrix z of shape $n-1$ (number of iterations) by 4:
- The rows represent iterations.
- First and second columns ($c1$ and $c2$ above): indexes of the clusters being merged.
- Third column ($\text{distance}(c1, c2)$): the distance between the clusters being merged.
- Fourth column (# observations): the number of examples in the newly formed cluster.

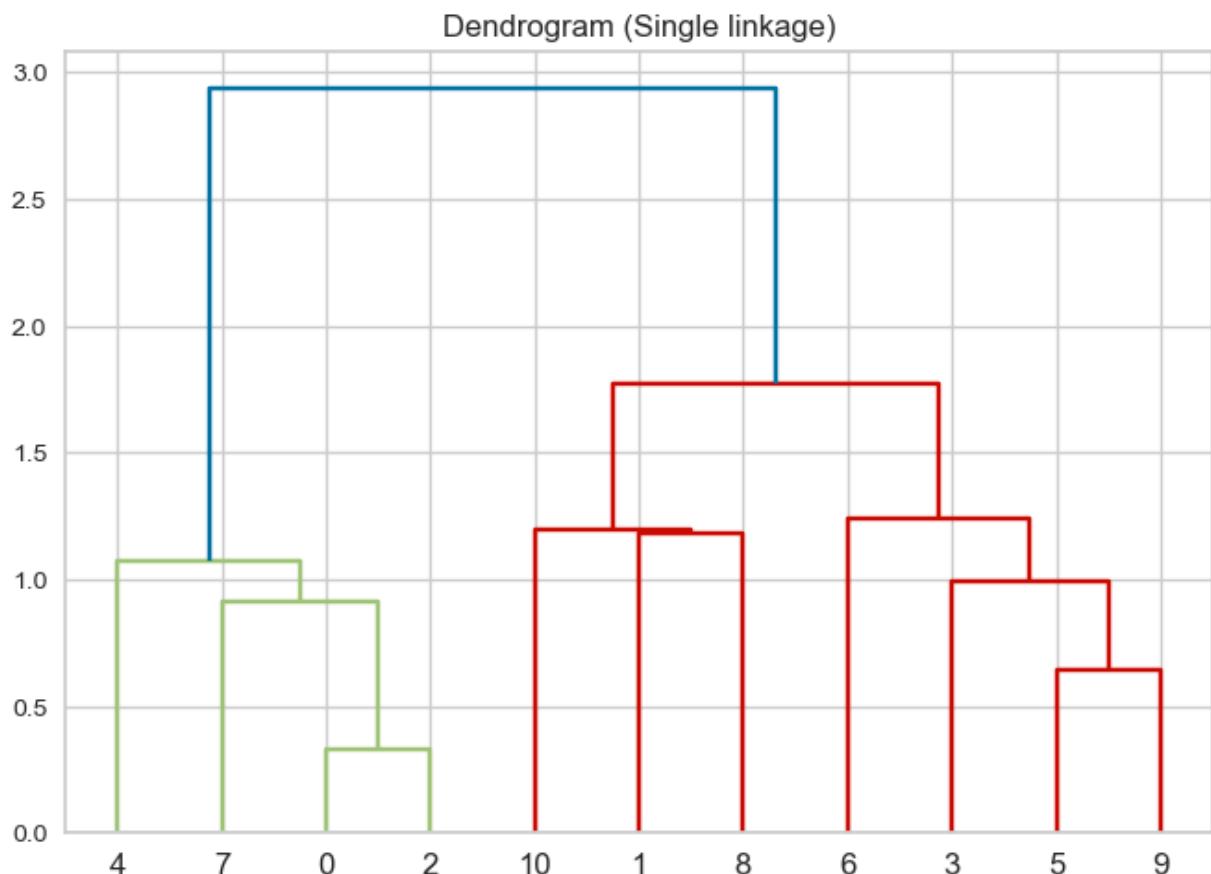
Creating dendrogram with single linkage

In [25]:

```

1 dendrogram(Z)
2 # Z is our single linkage matrix
3 plt.title("Dendrogram (Single linkage)");

```

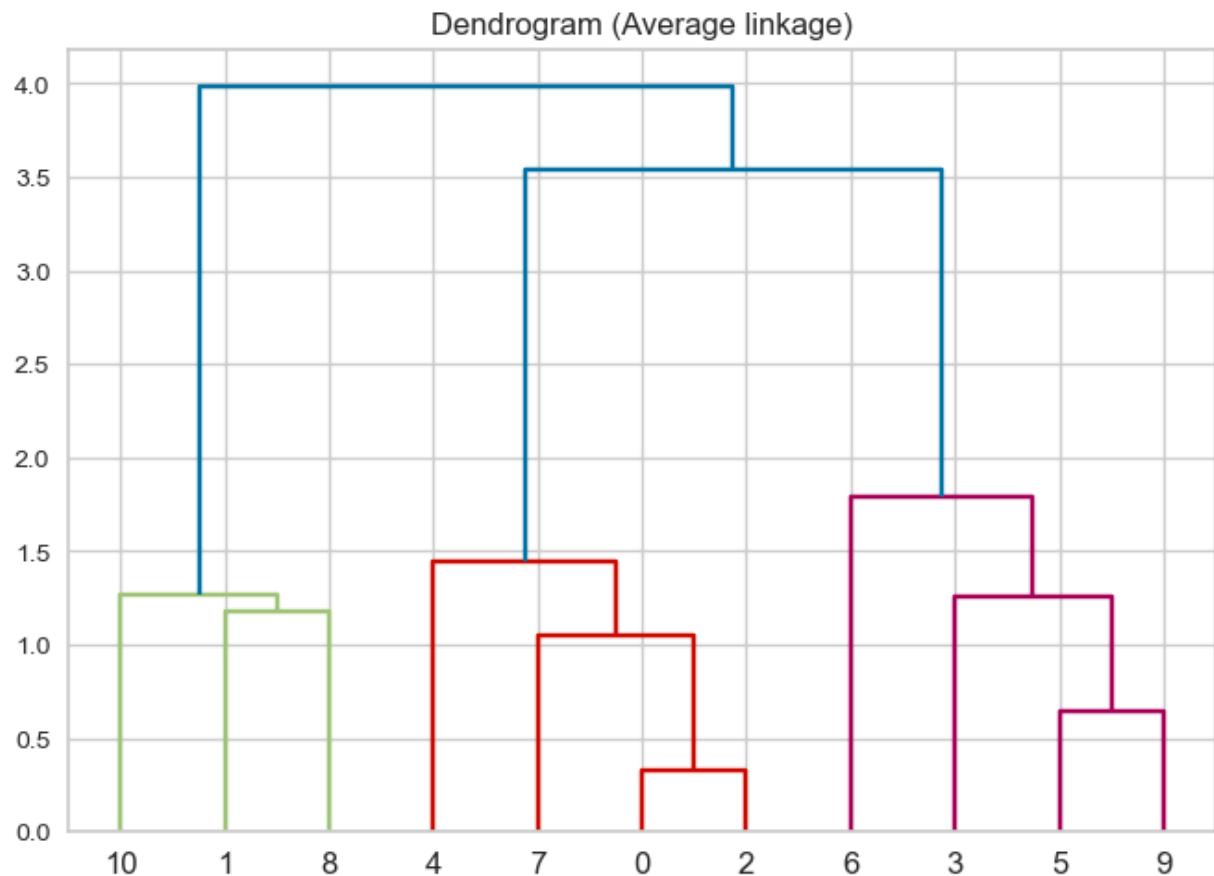


average linkage

- Merges two clusters that have the smallest average distance between all their points.
- `scipy.cluster.hierarchy's` `average` method gives us matrix z with the merging information using average linkage.

In [26]:

```
1 Z = average(X)
2 dendrogram(Z)
3 # Dendrogram with average linkage
4 plt.title("Dendrogram (Average linkage)");
```

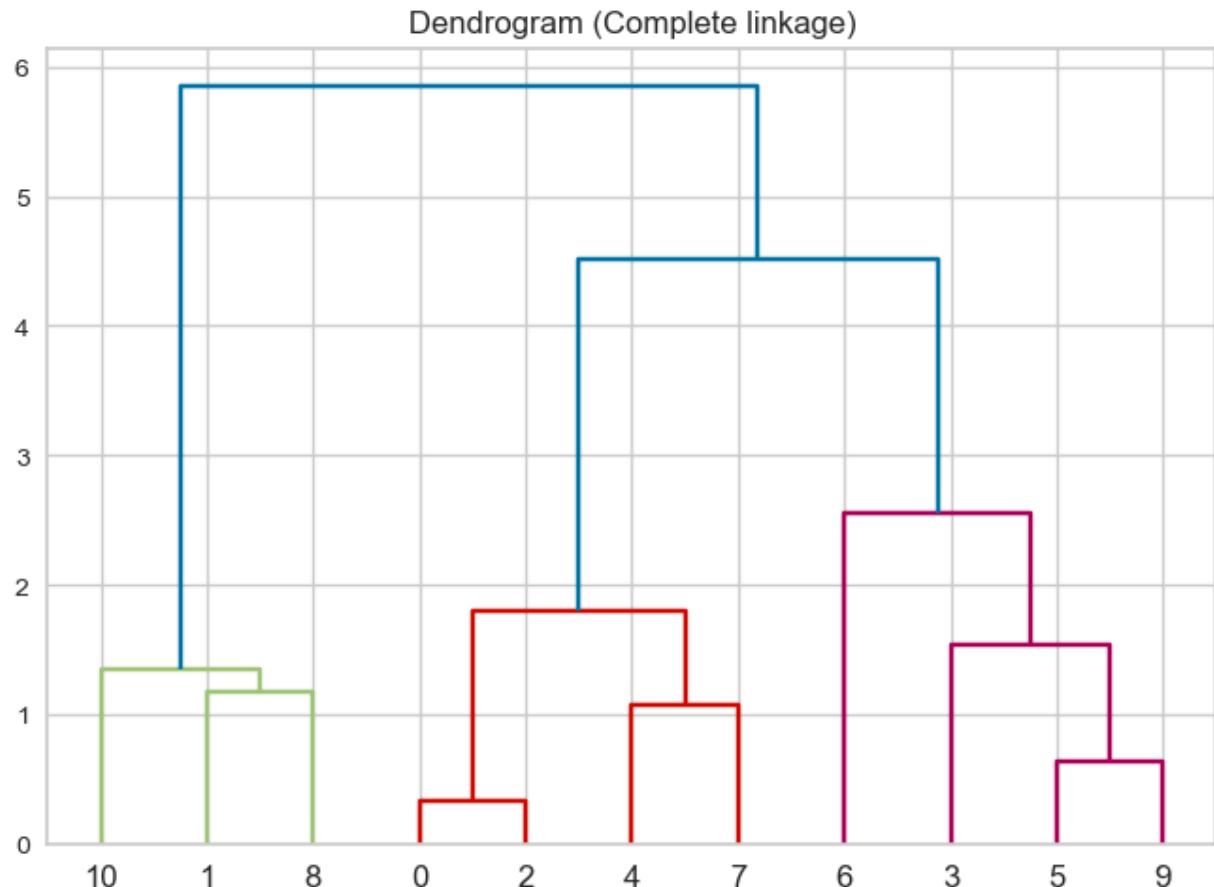


complete linkage

- Merges two clusters that have the smallest maximum distance between their points.

In [27]:

```
1 z = complete(X)
2 dendrogram(z)
3 # Dendrogram with complete linkage
4 plt.title("Dendrogram (Complete linkage)");
```

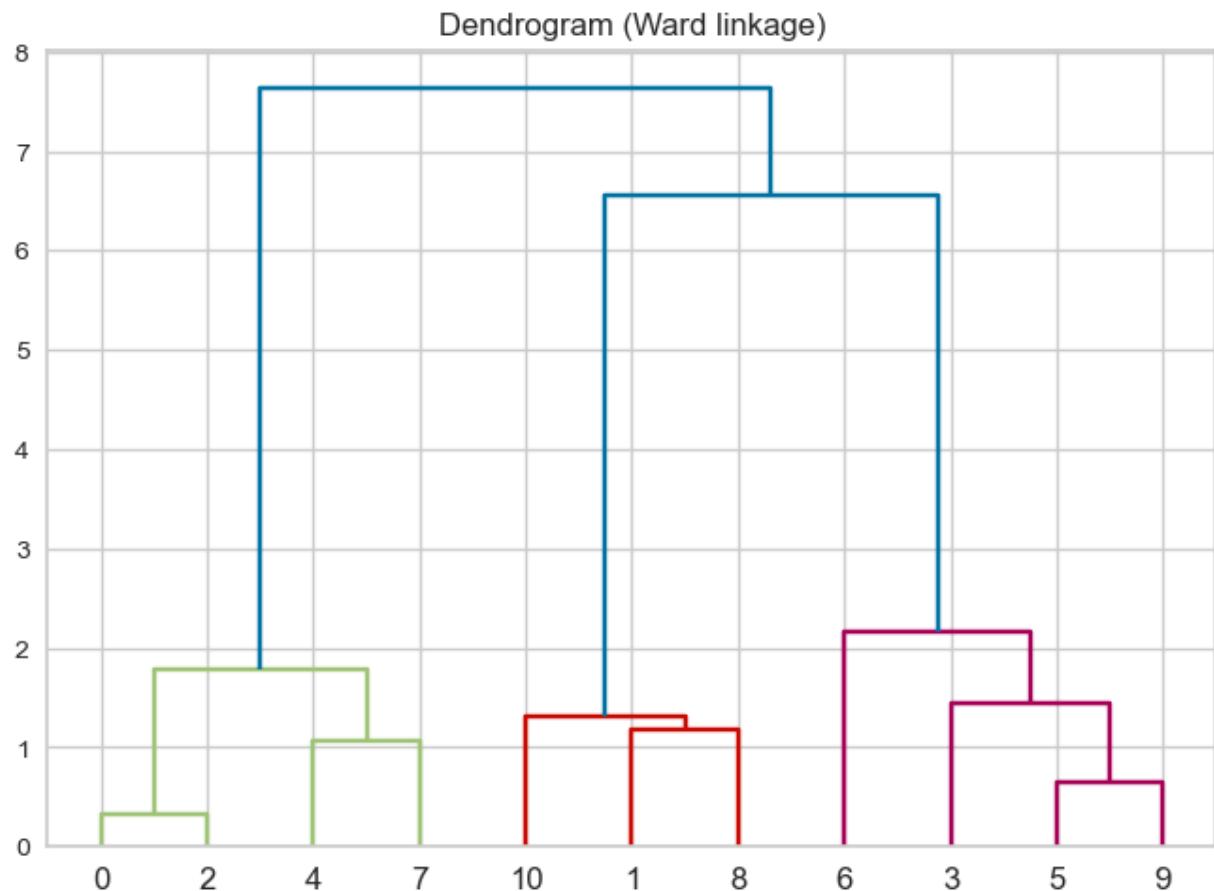


ward linkage

- Picks two clusters to merge such that the variance within all clusters increases the least.
- Often leads to equally sized clusters.

In [28]:

```
1 Z = ward(X)
2 dendrogram(Z)
3 # Dendrogram with ward linkage
4 plt.title("Dendrogram (Ward linkage)");
```



Hierarchical clustering on UN Subvotes dataset

- Let's use a dataset of votes on UN resolutions:

In [29]:

```

1 votes_df = pd.read_csv("../data/subvotes.csv")
2 votes = votes_df.pivot(index="country", columns="rcid")
3 votes = votes[np.sum(np.isnan(votes), axis=1) < 1]
4 print(votes.shape)
5 votes.head()

```

(17, 368)

Out[29]:

	rcid	2491	2492	2497	2504	2510	2526	2563	2610	2641	2645	...	5321	5333	5337	5358
	country															
Australia	1.0	2.0	1.0	3.0	3.0	3.0	1.0	1.0	1.0	1.0	1.0	...	1.0	1.0	1.0	1.0
Austria	1.0	1.0	1.0	2.0	2.0	2.0	1.0	1.0	1.0	1.0	1.0	...	1.0	1.0	1.0	1.0
Brazil	1.0	1.0	1.0	1.0	1.0	2.0	2.0	3.0	1.0	1.0	1.0	...	1.0	1.0	1.0	1.0
Colombia	1.0	1.0	1.0	1.0	1.0	2.0	1.0	2.0	1.0	1.0	1.0	...	1.0	1.0	1.0	1.0
Denmark	1.0	1.0	1.0	2.0	2.0	3.0	1.0	1.0	1.0	1.0	1.0	...	1.0	1.0	1.0	1.0

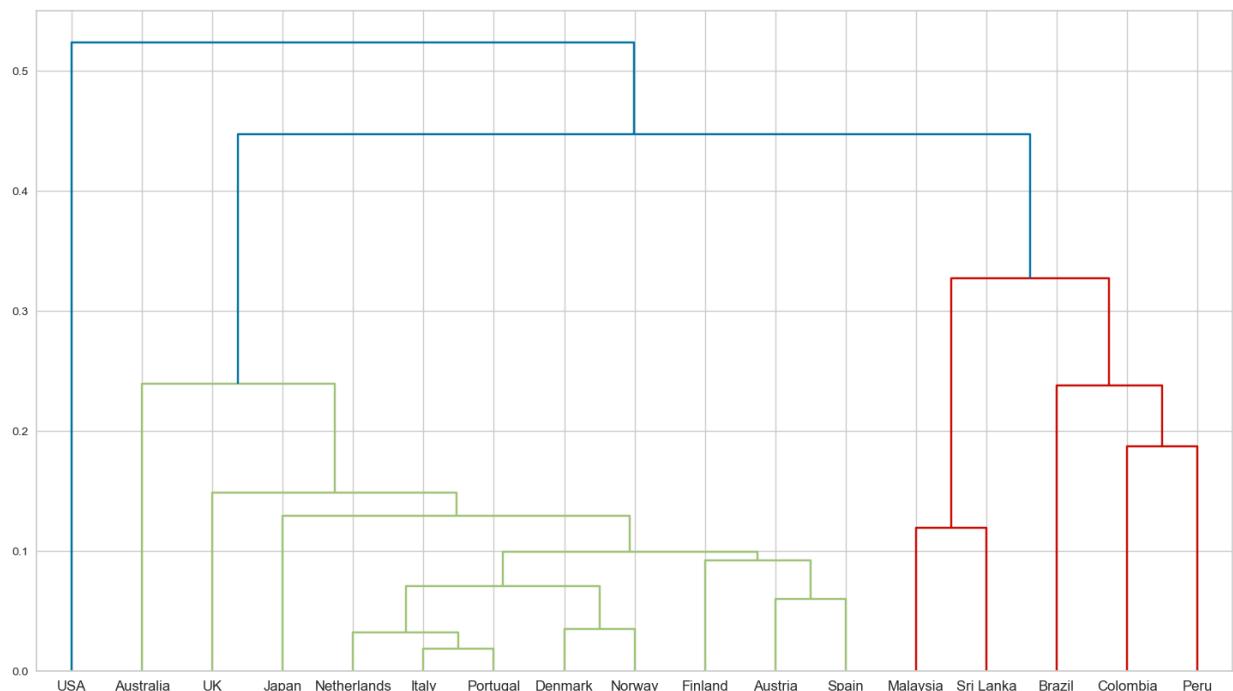
5 rows × 368 columns

- We have 17 countries and 368 votes.
- Let's cluster countries based on how they vote.

- We'll use [hamming distance \(\[https://en.wikipedia.org/wiki/Hamming_distance\]\(https://en.wikipedia.org/wiki/Hamming_distance\)\)](https://en.wikipedia.org/wiki/Hamming_distance) here because we are interested in knowing whether the countries agreed or disagreed on resolutions.

In [30]:

```
1 Z = linkage(votes, method="average", metric="hamming")
2 fig, ax = plt.subplots(figsize=(18, 10))
3 dendrogram(Z, p=6, ax=ax, labels=votes.index);
```



(Optional) Truncation

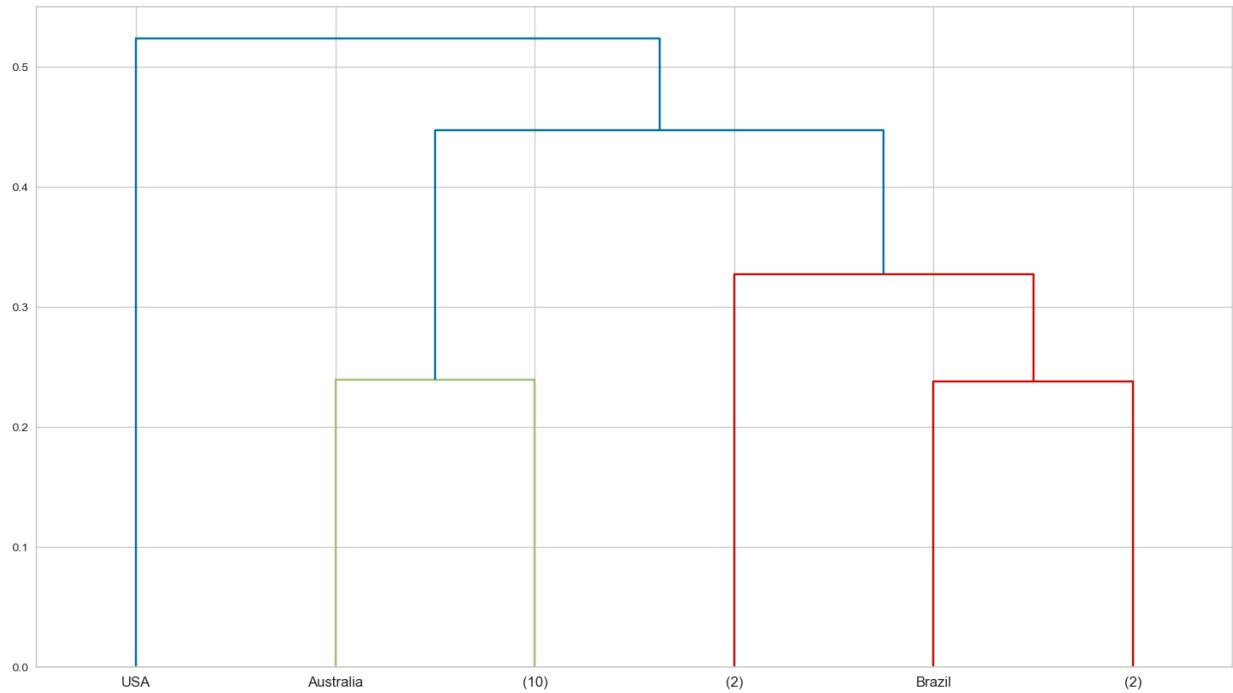
- If you want to truncate the tree in the dendrogram (specially when you have a big n) you can use the `truncate_mode`.

In [31]:

```

1 Z = linkage(votes, method="average", metric="hamming")
2 fig, ax = plt.subplots(figsize=(18, 10))
3 dendrogram(Z, p=6, truncate_mode="lastp", ax=ax, labels=votes.index);
4 # p is the number of leaves when truncate mode is "lastp"

```



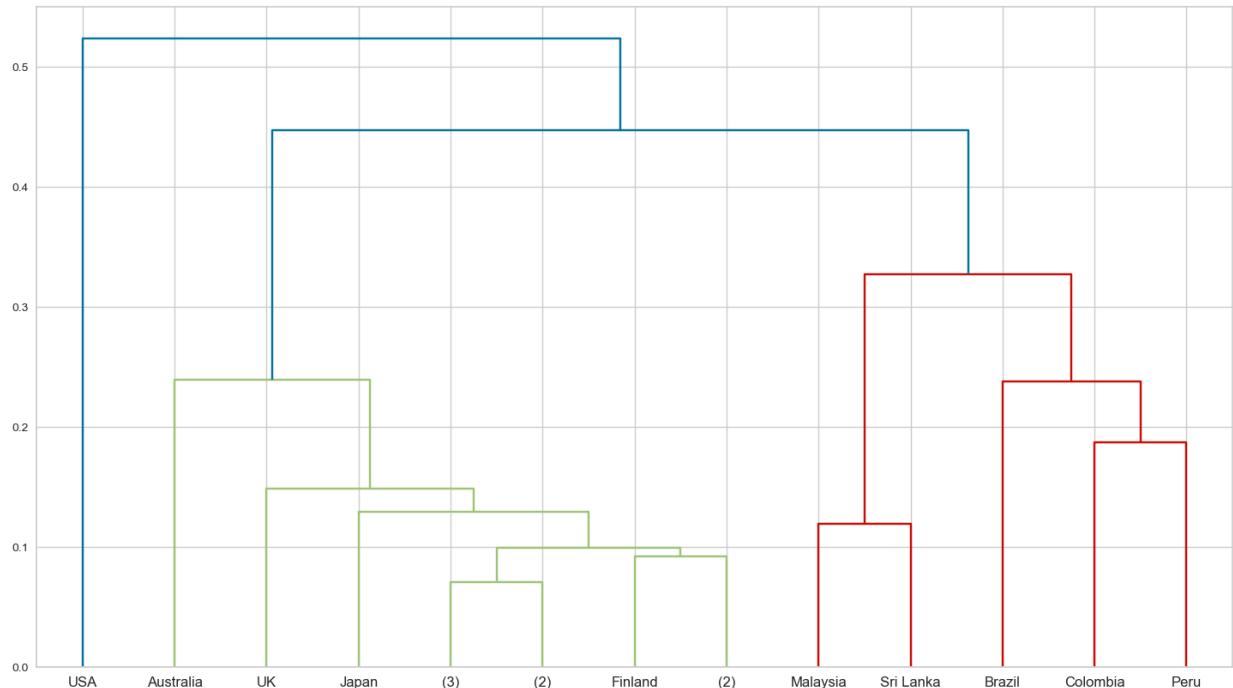
- Alternatively, you can truncate the tree down to p levels from the single cluster:

In [32]:

```

1 fig, ax = plt.subplots(figsize=(18, 10))
2 dendrogram(Z, p=6, truncate_mode="level", ax=ax, labels=votes.index);
3 # p is the max depth of the tree when truncate_mode is "level"

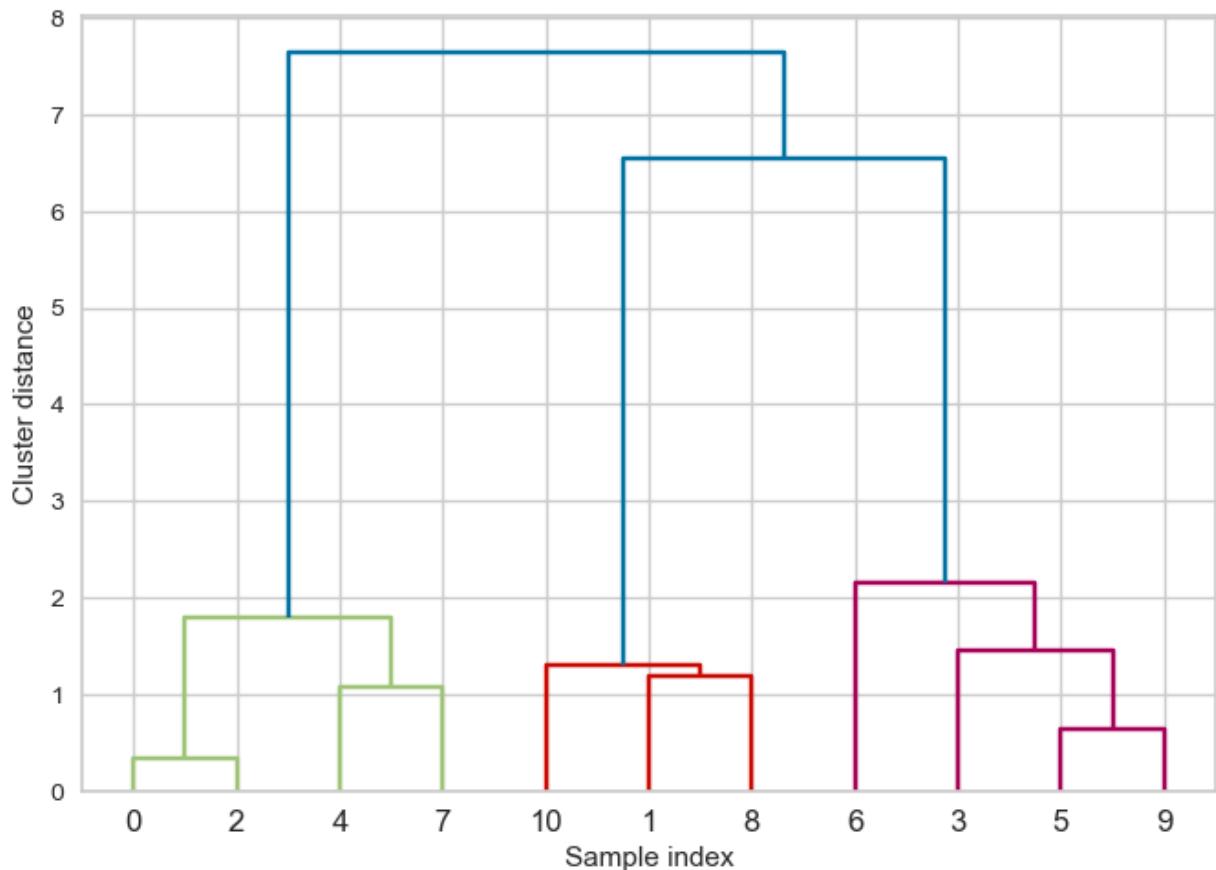
```



Let's go back to our toy dataset to understand truncation better.

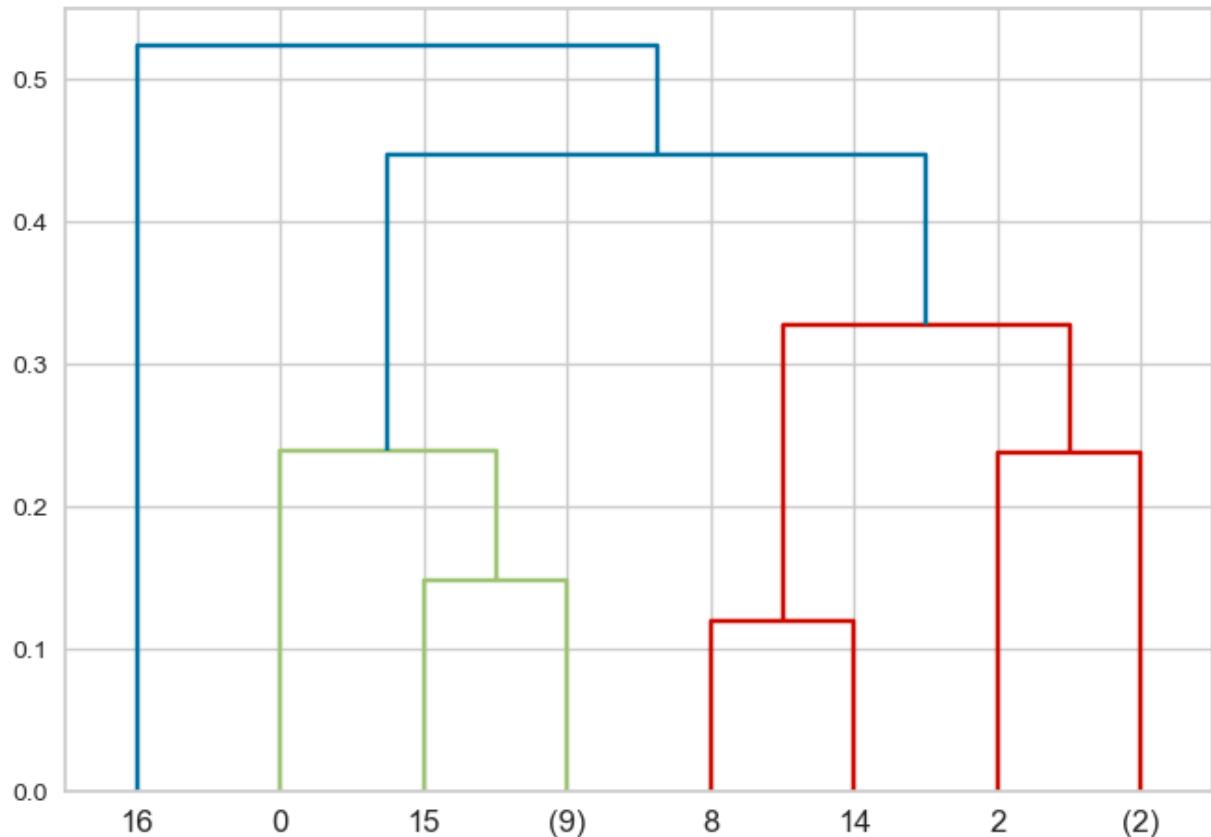
In [33]:

```
1 ax = plt.gca()
2 dendrogram(linkage_array, ax=ax)
3 plt.xlabel("Sample index")
4 plt.ylabel("Cluster distance");
```



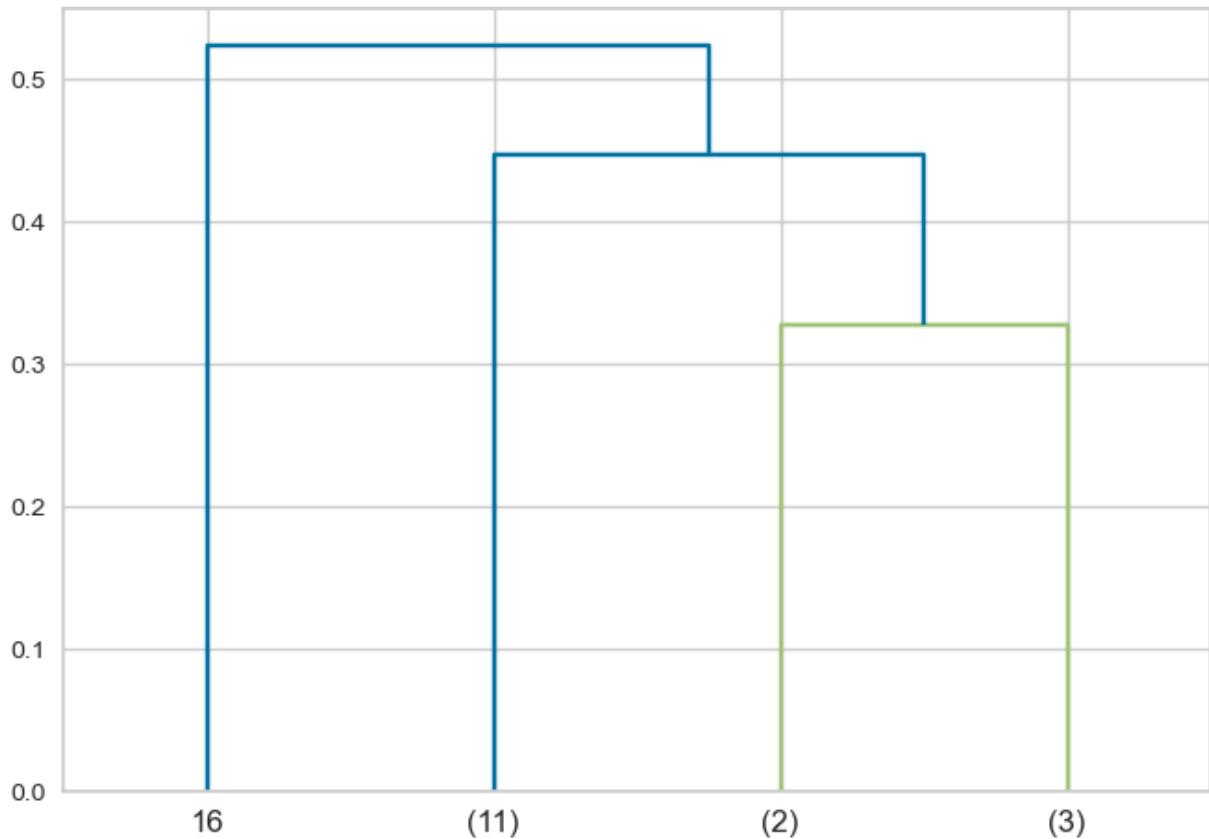
In [34]:

```
1 dendrogram(Z, p=3, truncate_mode="level");
2 # p is the max depth of the tree
```



In [35]:

```
1 dendrogram(Z, p=4, truncate_mode="lastp");
2 # p is the number of leaf nodes
```



Flat cluster

- To bring the clustering to a "flat" format, we can use `fcluster`

In [36]:

```
1 from scipy.cluster.hierarchy import fcluster
2
3 cluster_labels = fcluster(Z, 6, criterion="maxclust")
```

```
In [37]: 1 pd.DataFrame(cluster_labels, votes.index)
```

Out[37]: 0

country	
Australia	2
Austria	1
Brazil	5
Colombia	4
Denmark	1
Finland	1
Italy	1
Japan	1
Malaysia	3
Netherlands	1
Norway	1
Peru	4
Portugal	1
Spain	1
Sri Lanka	3
UK	1
USA	6

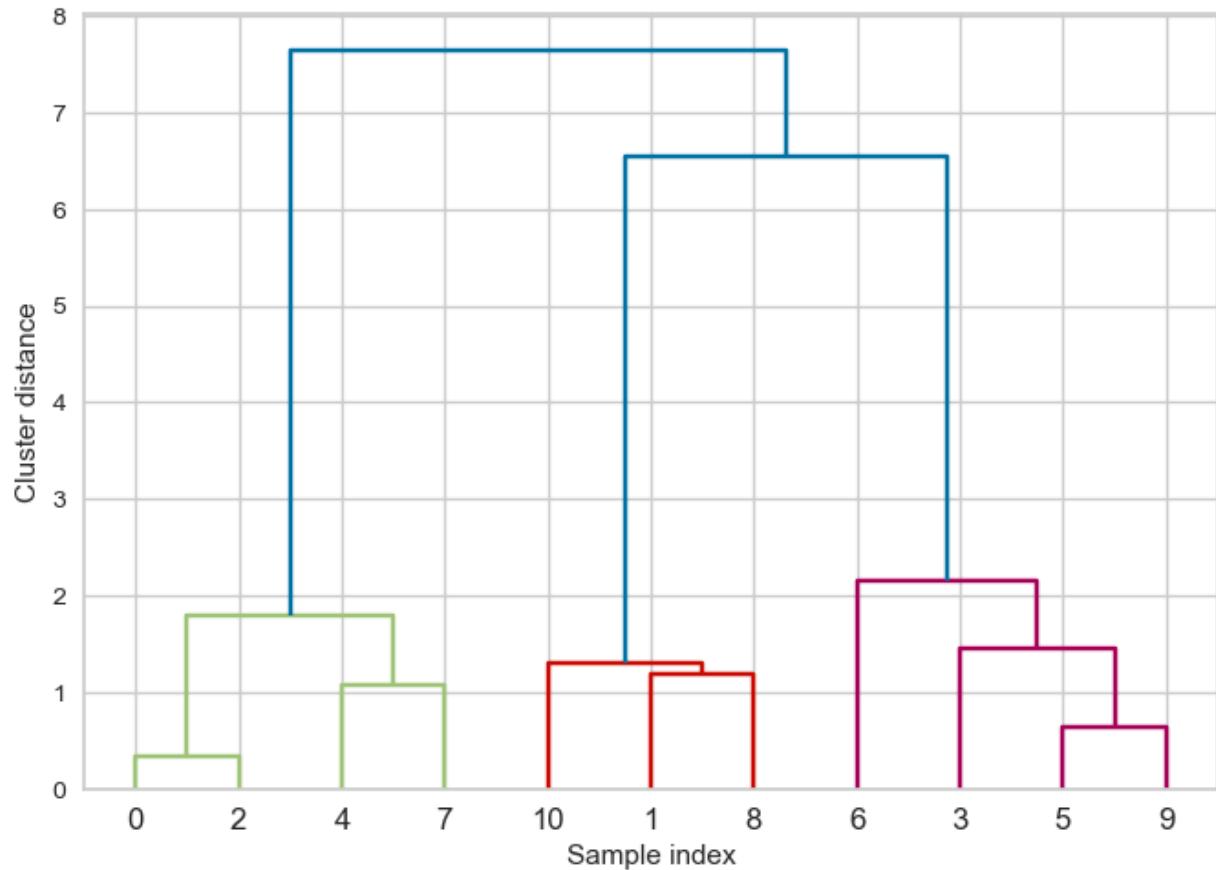
To understand this better, let's try it out on our toy dataset.

In [38]:

```

1 ax = plt.gca()
2 dendrogram(linkage_array, ax=ax)
3 plt.xlabel("Sample index")
4 plt.ylabel("Cluster distance");

```



In [39]:

```

1 cluster_labels = fcluster(linkage_array, 3, criterion="maxclust")
2 pd.DataFrame(cluster_labels, columns=[ "Cluster"])

```

Out[39]:

	Cluster
0	1
1	2
2	1
3	3
4	1
5	3
6	3
7	1
8	2
9	3
10	2

	Cluster
0	1
1	2
2	1
3	3
4	1
5	3
6	3
7	1
8	2
9	3
10	2

? ? Questions for you

Which statements are TRUE?

- (A) With tiny epsilon (`eps` in `sklearn`) and `min samples=1` (`min_samples=1` in `sklearn`) we are likely to end up with each point in its own cluster.
- (B) With a smaller value of `eps` and larger number for `min_samples` we are likely to end up with a one big cluster.
- (C) K-Means is more susceptible to outliers compared to DBSCAN.
- (D) In DBSCAN to be part of a cluster, each point must have at least `min_samples` neighbours in a given radius (including itself).
- (E) In DBSCAN, it is generally a good idea to run DBSCAN with a large number of different random orderings of training examples.

Which statements are TRUE?

- (A) In hierarchical clustering we do not have to worry about initialization.
- (B) Hierarchical clustering can only be applied to smaller datasets because dendograms are hard to visualize for large datasets.
- (C) In all the three clustering methods we saw (K-Means, DBSCAN, hierarchical clustering), there is a way to decide the granularity of clustering (i.e., how many clusters to pick).
- (D) To get robust clustering we can naively ensemble cluster labels (e.g., pick the most popular label) produced by different clustering methods.
- (E) If you have a high Silhouette score and very clean and robust clusters, it means that the algorithm has captured the semantic meaning in the data of our interest.

Applying clustering on face images

- We'll be working with `sklearn`'s [Labeled Faces in the Wild dataset \(`https://scikit-learn.org/0.16/datasets/labeled_faces.html`\)](https://scikit-learn.org/0.16/datasets/labeled_faces.html).
- The dataset has images of celebrities from the early 2000s downloaded from the internet.

Credit: This example is based on the example from [here](https://learning.oreilly.com/library/view/introduction-to-machine/9781449369880/ch03.html)

```
In [40]: 1 import matplotlib as mpl
2 from sklearn.datasets import fetch_lfw_people
3
4 mpl.rcParams.update(mpl.rcParamsDefault)
5 plt.rcParams["image.cmap"] = "gray"
```

Example images from the dataset

```
In [41]: 1 people = fetch_lfw_people(min_faces_per_person=20, resize=0.7)
2
3 fig, axes = plt.subplots(2, 5, figsize=(10, 5), subplot_kw={"xticks": (
4 for target, image, ax in zip(people.target, people.images, axes.ravel())
5     ax.imshow(image)
6     ax.set_title(people.target_names[target]))}
```

```
In [42]: 1 plt.show()
```



Winona Ryder



Jean Chretien



Carlos Menem



Ariel Sharon



Alvaro Uribe



Colin Powell



Recep Tayyip Erdogan



Gray Davis



George Robertson



Silvio Berlusconi

```
In [43]: 1 image_shape = people.images[0].shape
2 print("people.images.shape: {}".format(people.images.shape))
3 print("Number of classes: {}".format(len(people.target_names)))
```

people.images.shape: (3023, 87, 65)
Number of classes: 62

There are 3,023 images stored as arrays of 5655 pixels (87 by 65), of 62 different people:

```
In [44]: 1 counts = np.bincount(people.target) # count how often each target appears
          2 df = pd.DataFrame(counts, columns=["count"], index=people.target_names)
          3 df.sort_values("count", ascending=False)
```

Out[44]:

	count
George W Bush	530
Colin Powell	236
Tony Blair	144
Donald Rumsfeld	121
Gerhard Schroeder	109
...	...
Angelina Jolie	20
Jiang Zemin	20
Paul Bremer	20
Igor Ivanov	20
Michael Bloomberg	20

62 rows × 1 columns

Let's make the data less skewed by taking only 20 images of each person.

```
In [45]: 1 mask = np.zeros(people.target.shape, dtype=bool)
          2 for target in np.unique(people.target):
          3     mask[np.where(people.target == target)[0][:20]] = 1
          4
          5 X_people = people.data[mask]
          6 y_people = people.target[mask]
```

In [46]: 1 X_people.shape

Out[46]: (1240, 5655)

Representation of images

- Representation of input is very important when you cluster examples.
- In this example, we'll use PCA representation. (We won't learn about PCA in this course. You can think of it as a method to extract most important information from the given data.)

```
In [47]: 1 from sklearn.decomposition import PCA
          2
          3 pca = PCA(n_components=100, whiten=True, random_state=0)
          4 X_pca = pca.fit_transform(X_people)
```

Clustering faces with K-Means

We'll cluster the images with this new representation.

```
In [48]: 1 km = KMeans(n_clusters=10, random_state=10)
2 km.fit(X_pca)
```

Out[48]: KMeans(n_clusters=10, random_state=10)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

What are the sizes of the clusters?

```
In [49]: 1 labels_km = km.fit_predict(X_pca)
2 print("Cluster sizes k-means: {}".format(np.bincount(labels_km)))
```

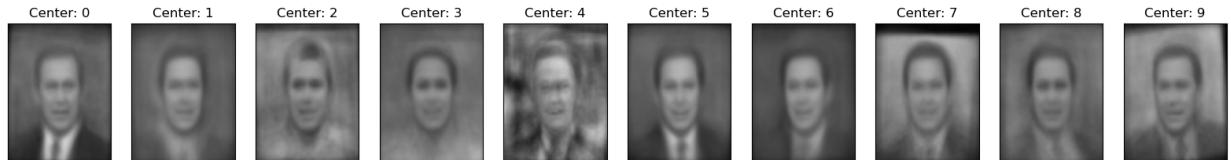
Cluster sizes k-means: [79 243 41 125 3 207 279 75 125 63]

Let's examine cluster centers. Are they going to be real images from the dataset?

```
In [50]: 1 def get_cluster_images(km, X_people, y_people, target_names, X_pca=None):
2     image_shape = (87, 65)
3     fig, axes = plt.subplots(1, 6, subplot_kw={'xticks': (), 'yticks': (),
4                                         figsize=(10, 10), gridspec_kw={"hspace": .5})
5     center = km.cluster_centers_[cluster]
6
7     mask = km.labels_ == cluster
8
9     if pca:
10        dists = np.sum((X_pca - center) ** 2, axis=1)
11        dists[~mask] = np.inf
12        inds = np.argsort(dists)[:5]
13        axes[0].imshow(pca.inverse_transform(center).reshape(image_shape))
14    else:
15        dists = np.sum((X_people - center) ** 2, axis=1)
16        dists[~mask] = np.inf
17        inds = np.argsort(dists)[:5]
18        axes[0].imshow(center.reshape(image_shape), vmin=0, vmax=1)
19
20    axes[0].set_title('Cluster center %d' % (cluster))
21    i = 1
22    for image, label in zip(X_people[inds], y_people[inds]):
23        axes[i].imshow(image.reshape(image_shape), vmin=0, vmax=1)
24        axes[i].set_title("%s" % (target_names[label].split()[-1]), fontweight='bold')
25        i += 1
26    plt.show()
```

In [51]:

```
1 plot_faces_cluster_centers(km, pca)
```



- The centers found by K-Means are smooth versions of faces which makes sense.
- Intuitively, they seem to capture some interesting characteristics of faces:
 - grumpy faces
 - faces somewhat rotated to the left/right
 - Smiley faces

Let's examine images for different centers.

In [52]:

```
1 get_cluster_images(
2     km, X_people, y_people, people.target_names, pca=pca, X_pca=X_pca,
3 )
```

Cluster center 4



Rumsfeld



Capriati



Jolie



Agassi



Agassi

In [53]:

```
1 get_cluster_images(
2     km, X_people, y_people, people.target_names, pca=pca, X_pca=X_pca,
3 )
```

Cluster center 8



Abbas



Jolie



Menem



Agassi



Ryder

Clustering faces with DBSCAN

In [54]:

```
1 dbSCAN = DBSCAN()  
2 labels = dbSCAN.fit_predict(X_pca)  
3 print("Unique labels: {}".format(np.unique(labels)))
```

Unique labels: [-1]

With default hyperparameters, we get all points as noise points.

Tuning eps

In [55]:

```
1 people.target_names[y_people].shape
```

Out[55]:

Let's examine at distances between images.

In [56]:

```

1 from sklearn.metrics.pairwise import euclidean_distances
2
3 dists = euclidean_distances(X_pca)
4 np.fill_diagonal(dists, np.inf)
5
6 dist_df = pd.DataFrame(
7     dists, index=people.target_names[y_people], columns=people.target_n
8 )
9
10 dist_df.iloc[10:20, 10:20]

```

Out[56]:

	George W Bush	George W Bush	Nestor Kirchner	Jean Chretien	Bill Clinton	George W Bush	Carlos Menem	Alvaro Uribe	G
George W Bush	inf	13.494180	12.842854	15.728106	15.821581	13.411802	15.145203	16.356001	1
George W Bush	13.494180	inf	11.172112	15.952421	16.520470	12.882513	14.535444	15.917523	1
Nestor Kirchner	12.842854	11.172112	inf	13.967069	14.954653	9.952094	12.282001	13.954332	
Jean Chretien	15.728106	15.952421	13.967069	inf	17.792767	15.888998	16.187300	17.894358	1
Bill Clinton	15.821581	16.520470	14.954653	17.792767	inf	15.382248	17.404568	16.727720	1
George W Bush	13.411802	12.882513	9.952094	15.888998	15.382248	inf	14.127582	15.053101	
Carlos Menem	15.145203	14.535444	12.282001	16.187300	17.404568	14.127582	inf	15.061512	1
Alvaro Uribe	16.356001	15.917523	13.954332	17.894358	16.727720	15.053101	15.061512	inf	1
George W Bush	11.605841	11.627776	8.903399	15.051611	14.883553	9.760360	11.510704	13.337687	
Colin Powell	12.597251	12.644590	12.063414	15.960009	15.612200	12.570139	14.349072	14.868257	1

```
In [57]: 1 dist_df.describe()
```

Out[57]:

	Winona Ryder	Jean Chretien	Carlos Menem	Ariel Sharon	Alvaro Uribe	Colin Powell	Recep Tayyip Erdogan
count	1240.000000	1240.000000	1240.000000	1240.000000	1240.000000	1240.000000	1240.000000
mean	inf	inf	inf	inf	inf	inf	inf
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	10.376113	8.781528	7.795739	11.128681	10.745722	8.586018	6.987148
25%	13.499337	12.704505	11.151741	13.726987	14.150235	12.042804	10.153908
50%	14.547655	13.802588	12.324697	14.768760	15.066293	13.137733	11.481316
75%	15.606746	15.057733	13.601164	15.953466	16.179981	14.411716	12.877707
max	inf	inf	inf	inf	inf	inf	inf

8 rows × 1240 columns

In [58]:

```

1 for eps in [6, 7, 8, 9, 10, 11, 12, 14]:
2     print("\neps={}".format(eps))
3     dbSCAN = DBSCAN(eps=eps, min_samples=3)
4     labels = dbSCAN.fit_predict(X_pca)
5     print("Number of clusters: {}".format(len(np.unique(labels))))
6     print("Cluster sizes: {}".format(np.bincount(labels + 1)))

```

```

eps=6
Number of clusters: 2
Cluster sizes: [1236    4]

eps=7
Number of clusters: 2
Cluster sizes: [1181    59]

eps=8
Number of clusters: 3
Cluster sizes: [1036   201     3]

eps=9
Number of clusters: 3
Cluster sizes: [816  420     4]

eps=10
Number of clusters: 2
Cluster sizes: [577  663]

eps=11
Number of clusters: 2
Cluster sizes: [349  891]

eps=12
Number of clusters: 2
Cluster sizes: [ 166 1074]

eps=14
Number of clusters: 2
Cluster sizes: [  21 1219]

```

- For lower `eps` all images are labeled as noise.
- For `eps=7` we get many noise points and many small clusters.
- For `eps=8` and `eps=9` we get many noise points but we also get one large cluster and a few smaller clusters.
- Starting `eps=10` we get one big cluster and noise points.
- There is never more than one large cluster suggesting that all the images are more or less equally similar/dissimilar to the rest.

Noise images identified by DBSCAN

```
In [59]: 1 dbSCAN = DBSCAN(eps=14, min_samples=3)
2 labels = dbSCAN.fit_predict(X_pca)
3 print("Unique labels: {}".format(np.unique(labels)))
```

Unique labels: [-1 0]

```
In [60]: 1 print_dbSCAN_noise_images(X_people, y_people, dbSCAN, labels)
```

```
In [61]: 1 plt.show()
```



- We can guess why these images are noise images. There are odd angles, cropping, sun glasses, hands near faces etc.

Let's examine DBSCAN clusters.

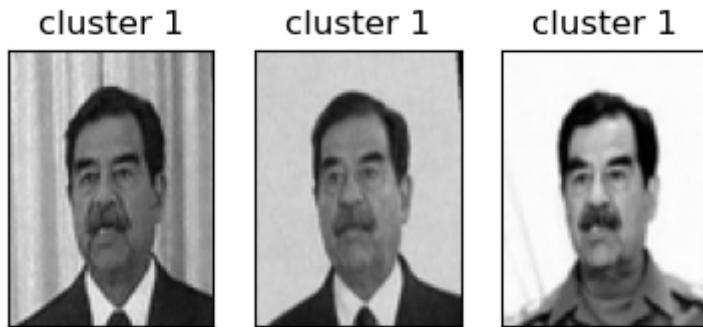
```
In [62]: 1 dbSCAN = DBSCAN(min_samples=3, eps=8)
2 labels = dbSCAN.fit_predict(X_pca)
3 print("Number of clusters: {}".format(len(np.unique(labels))))
4 print("Cluster sizes: {}".format(np.bincount(labels + 1)))
```

Number of clusters: 3
Cluster sizes: [1036 201 3]

- Some clusters correspond to people with distinct faces and facial expressions.
- It's also capturing orientation of the face.

```
In [63]: 1 print_dbSCAN_clusters(X_people, y_people, labels)
```

```
In [64]: 1 plt.show()
```



(Optional) Clustering faces with hierarchical clustering

Let's examine the dendrogram.

```
In [65]: 1 z = ward(X_pca)
2 plt.figure(figsize=(20, 15))
3 dendrogram(z, p=7, truncate_mode="level", no_labels=True)
4 plt.xlabel("Sample index")
5 plt.ylabel("Cluster distance");
```

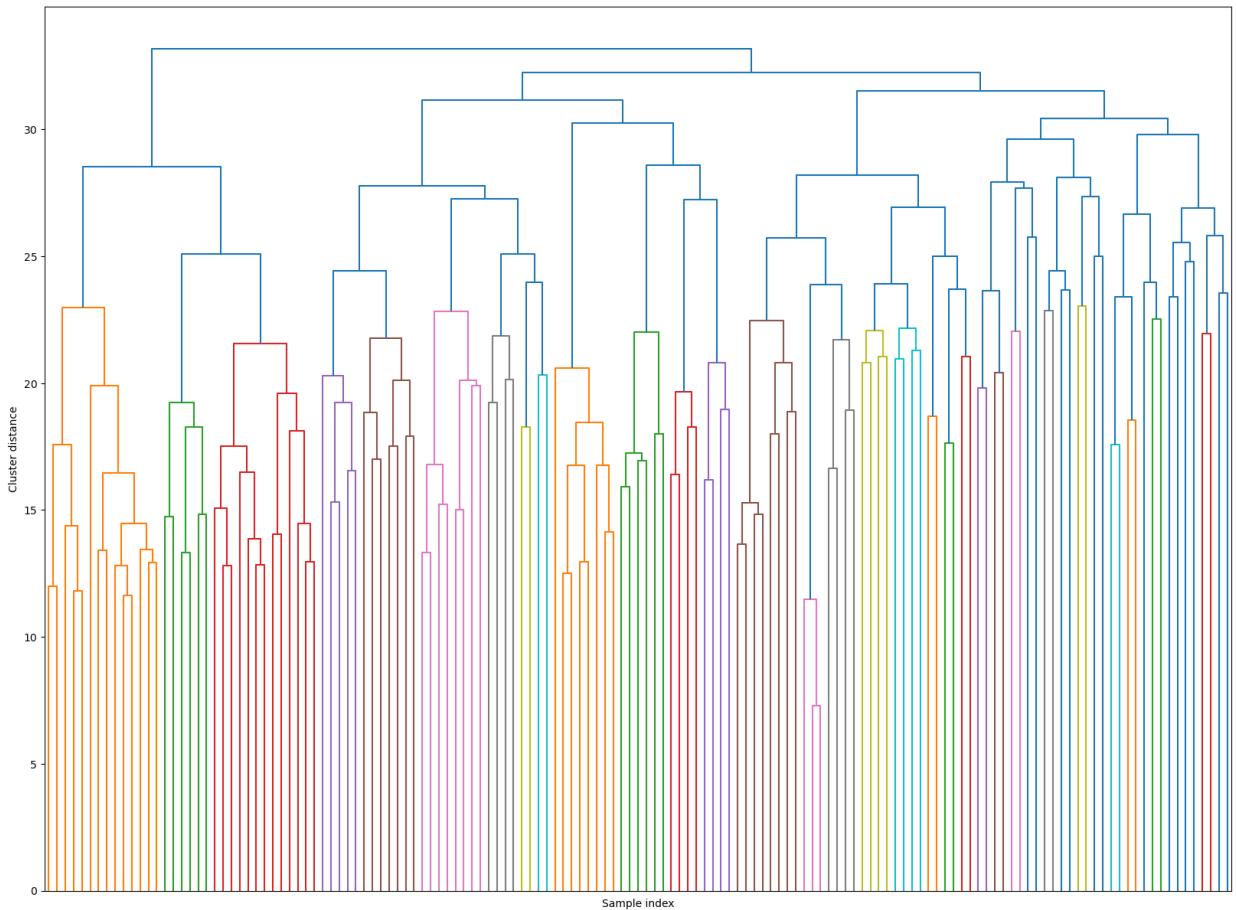
```
In [66]: 1 cluster_labels = fcluster(z, 40, criterion="maxclust") # let's get fla
```

In [67]:

```

1 hand_picked_clusters = [2, 3, 6, 29, 30, 36, 38]
2 print_hierarchical_clusters(
3     X_people, y_people, people.target_names, cluster_labels, hand_picked_clusters)
4 )
5 plt.show()

```





Final comments, summary, and reflection

Take-home message

- We saw three methods for clustering: K-Means, DBSCAN, and hierarchical clustering.
- There are many more clustering algorithms out there which we didn't talk about. For example see [this overview of clustering methods \(<https://scikit-learn.org/stable/modules/clustering.html#overview-of-clustering-methods>\)](https://scikit-learn.org/stable/modules/clustering.html#overview-of-clustering-methods).
- Two important aspects of clustering
 - Choice of distance metric
 - Data representation
- Choosing the appropriate number of clusters for a given problem is quite hard.
- A lot of manual interpretation is involved in clustering.

A few comments on clustering evaluation

- If you know the ground truth, you can use metrics such as [adjusted random score](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html) (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html) or [normalized mutual information score](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html) (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html).
- We can't use accuracy scores.
 - Because the labels themselves are meaningless in clustering.
- Usually ground truth is not available, and if it is available we would probably go with supervised models.

- The silhouette score works for different clustering methods and it can give us some intuition

A couple of ways to evaluate clustering:

- Using *robustness-based* clustering metrics
- The idea is to run a clustering algorithm or a number of clustering algorithms after adding some noise to the data or using different parameter settings and comparing outcomes.
- If many models, perturbations, and parameters are giving the same result, the clustering is likely to be trustworthy.
- That said, even though all clustering models give similar results, the clusters might not capture the aspect you are interested in.
- So you cannot really avoid manual analysis!!

Resources

- Check out this nice comparison of [sklearn clustering algorithms](https://scikit-learn.org/stable/modules/clustering.html#overview-of-clustering-methods) (<https://scikit-learn.org/stable/modules/clustering.html#overview-of-clustering-methods>).
- [DBSCAN Visualization](https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/) (<https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/>)
- [Clustering with Scikit with GIFs](https://dashee87.github.io/data%20science/general/Clustering-with-Scikit-with-GIFs/) (<https://dashee87.github.io/data%20science/general/Clustering-with-Scikit-with-GIFs/>)

CPSC 330

Applied Machine Learning

Lecture 16: Recommender Systems

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

```
In [1]: 1 import os
2 import random
3 import sys
4 import time
5
6 import numpy as np
7 import pandas as pd
8
9 sys.path.append("../code/.")
10 import matplotlib.pyplot as plt
11 from plotting_functions import *
12 from plotting_functions_unsup import *
13 from sklearn.decomposition import PCA
14 from sklearn.model_selection import cross_validate, train_test_split
15 from sklearn.preprocessing import StandardScaler
16
17 plt.rcParams["font.size"] = 16
18 import matplotlib.cm as cm
19
20 # plt.style.use("seaborn")
21 %matplotlib inline
22 pd.set_option("display.max_colwidth", 0)
```

Learning outcomes

From this lecture, students are expected to be able to:

- State the problem of recommender systems.
- Describe components of a utility matrix.
- Create a utility matrix given ratings data.
- Describe a common approach to evaluate recommender systems.
- Implement some baseline approaches to complete the utility matrix.
- Explain the idea of collaborative filtering.

- Explain some serious consequences of recommendation systems.

Announcements

- Homework 7 is due Wednesday March 22, 11:59pm.

Recommender systems motivation

What is a recommender system?

- A recommender or a recommendation system **recommends** a particular product or service to users they are likely to consume.

The image displays three examples of recommender systems:

- Amazon:** Shows a product page for "The Reflective Educator's Guide to Classroom Research: Learning to..." by Nancy Fichtman Dana. It features a green oval around the text "Customers who bought this item also bought" which points to two other book covers: "The Reflective Educator's Guide to Classroom Research: Learning to..." and "Caring: A Relational Approach to Ethics and Moral Education".
- LinkedIn:** Shows a screenshot of the LinkedIn homepage with a green oval around the text "Jobs you may be interested in" which points to a section for "Any location · Any industry · 0 to 10,000+".
- Netflix:** Shows a screenshot of the Netflix homepage with a green oval around the text "Congratulations! Movies we think You will ❤️" which points to a grid of movie thumbnails including "Spider-Man", "300", "The Rundown", and "Bad Boys II".

Example: Recommender Systems

- A client goes to Amazon to buy products.
- Amazon has some information about the client. They also have information about other clients buying similar products.
- What should they recommend to the client, so that they buy more products?
- There's no "right" answer (no label).

- The whole idea is to understand user behavior in order to recommend them products they are

Why should we care about recommendation systems?

- Almost everything we buy or consume today is in some way or the other influenced by recommendation systems.
 - Music (Spotify), videos (YouTube), news, books and products (Amazon), movies (Netflix), jokes, restaurants, dating , friends (Facebook), professional connections (Linkedin)
- Recommendation systems are at the core of the success of many companies.
 - Amazon
 - [Netflix \(<https://help.netflix.com/en/node/100639>\)](https://help.netflix.com/en/node/100639)

Another example: [QxMD \(<https://qxmd.com/>\)](https://qxmd.com/)

- Present personalized journal article recommendations to health care professionals.

What kind of data do we need to build recommendation systems?

- **User ratings data** (most common)
- **Features related to items or users**
- Customer purchase history data

Main approaches

- Collaborative filtering
 - "Unsupervised" learning
 - We only have labels y_{ij} (or rating data, of user i for item j).
 - We learn features.
- Content-based recommenders
 - Supervised learning
 - Extract features x_i of users and/or items and building a model to predict rating y_i given x_i .
 - Apply model to predict for new users/items.
- Hybrid
 - Combining collaborative filtering with content-based filtering

The Netflix prize

The screenshot shows the official Netflix Prize website. At the top, there's a yellow banner with the words "Netflix Prize" and a large red "COMPLETED" stamp. Below the banner is a navigation bar with links: Home, Rules, Leaderboard, Update, and Download. The main section is titled "Leaderboard" in large blue letters. It says "Showing Test Score. [Click here to show quiz score](#)". There's a dropdown menu set to "Display top 20 leaders". The table below lists the top 8 teams, all of which achieved the Grand Prize RMSE of 0.8567. The winning team is BellKor's Pragmatic Chaos.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43

[Source \(<https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429>\)](https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429)

The Netflix prize

- 100M ratings from 0.5M users on 18k movies.
- Grand prize was \$1M for first team to reduce squared error at least by 10%.
- Winning entry (and most entries) used collaborative filtering:
 - Methods that only looks at ratings, not features of movies/users.
- A simple collaborative filtering method that does really well:
 - Now adopted by many companies.

Recommender systems problem

Problem formulation

- Most often the data for recommender systems come in as **ratings** for a set of items from a set of users.
- We have two entities: **N\$ users** and **M\$ items**.
- **Users** are consumers.
- **Items** are the products or services offered.
 - E.g., movies (Netflix), books (Amazon), songs (spotify), people (tinder)

					
	Item 1	Item 2	Item 3	Item 4	Item 5
User	?	?	2	?	3
User	3	?	?	?	?
User	?	5	4	?	5
User	?	?	?	?	?
User	?	?	?	5	?
User	?	5	4	3	?

Utility matrix

- A **utility matrix** is the matrix that captures **interactions** between **N\$ users** and **M\$ items**.
- The interaction may come in different forms:
 - ratings, clicks, purchases

					
	Item 1	Item 2	Item 3	Item 4	Item 5
User	?	?	2	?	3
User	3	?	?	?	?
User	?	5	4	?	5
User	?	?	?	?	?
User	?	?	?	5	?
User	?	5	4	3	?

Utility matrix

- Below is a toy utility matrix. Here $N\$ = 6$ and $M\$ = 5$.

- Each entry y_{ij} (\$i^{\text{th}}\$ row and \$j^{\text{th}}\$ column) denotes the rating given by the user \$i\$ to item \$j\$.
- We represent users in terms of items and items in terms of users.

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	?	?	2	?	3
User 2	3	?	?	?	?
User 3	?	5	4	?	5
User 4	?	?	?	?	?
User 5	?	?	?	5	?
User 6	?	5	4	3	?

Sparsity of utility matrix

- The utility matrix is very sparse because usually users only interact with a few items.
- For example:
 - all Netflix users will have rated only a small percentage of content available on Netflix
 - all amazon clients will have rated only a small fraction of items among all items available on Amazon

What do we predict?

Given a utility matrix of N users and M items, **complete the utility matrix**. In other words, **predict missing values in the matrix**.

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	?	?	2	?	3
User 2	3	?	?	?	?
User 3	?	5	4	?	5
User 4	?	?	?	?	?
User 5	?	?	?	5	?
User 6	?	5	4	3	?

- Once we have predicted ratings, we can recommend items to users they are likely to rate higher.

Example dataset: [Jester 1.7M jokes ratings dataset](https://www.kaggle.com/vikashrajluhaniwal/jester-17m-jokes-ratings-dataset?select=jester_ratings.csv)
[\(https://www.kaggle.com/vikashrajluhaniwal/jester-17m-jokes-ratings-dataset?select=jester_ratings.csv\)](https://www.kaggle.com/vikashrajluhaniwal/jester-17m-jokes-ratings-dataset?select=jester_ratings.csv)

- We'll use a sample of [Jester 1.7M jokes ratings dataset](https://www.kaggle.com/vikashrajluhaniwal/jester-17m-jokes-ratings-dataset) (<https://www.kaggle.com/vikashrajluhaniwal/jester-17m-jokes-ratings-dataset>) to demonstrate different recommendation systems.

The dataset comes with two CSVs

- A CSV containing ratings (-10.0 to +10.0) of 150 jokes from 59,132 users.
- A CSV containing joke IDs and the actual text of jokes.

Some jokes might be offensive. Please do not look too much into the actual text data if you are sensitive to such language.

- Recommendation systems are most effective when you have a large amount of data.
- But we are only taking a sample here for speed.

```
In [2]: 1 filename = "../data/jester_ratings.csv"
          2 ratings_full = pd.read_csv(filename)
          3 ratings = ratings_full[ratings_full["userId"] <= 4000]
```

```
In [3]: 1 ratings.head()
```

Out[3]:

	userId	jokeId	rating
0	1	5	0.219
1	1	7	-9.281
2	1	8	-9.281
3	1	13	-6.781
4	1	15	0.875

```
In [4]: 1 user_key = "userId"
          2 item_key = "jokeId"
```

Dataset stats

In [5]: 1 ratings.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 141362 entries, 0 to 141361
Data columns (total 3 columns):
 #   Column   Non-Null Count   Dtype  
--- 
 0   userId    141362 non-null   int64  
 1   jokeId   141362 non-null   int64  
 2   rating    141362 non-null   float64 
dtypes: float64(1), int64(2)
memory usage: 4.3 MB
```

In [6]: 1 def get_stats(ratings, item_key="jokeId", user_key="userId"):
2 print("Number of ratings: ", len(ratings))
3 print("Average rating: %0.3f" % (np.mean(ratings["rating"])))
4 N = len(np.unique(ratings[user_key]))
5 M = len(np.unique(ratings[item_key]))
6 print("Number of users (N): %d" % N)
7 print("Number of items (M): %d" % M)
8 print("Fraction non-nan ratings: %0.3f" % (len(ratings) / (N * M)))
9 return N, M
10
11
12 N, M = get_stats(ratings)

```
Number of ratings: 141362
Average rating: 1.200
Number of users (N): 3635
Number of items (M): 140
Fraction non-nan ratings: 0.278
```

Creating utility matrix

- Let's construct utility matrix with `number of users` rows and `number of items` columns from the `ratings` data.

Note we are constructing a non-sparse matrix for demonstration purpose here. In real life it's recommended that you work with sparse matrices.

In [7]: 1 user_mapper = dict(zip(np.unique(ratings[user_key]), list(range(N))))
2 item_mapper = dict(zip(np.unique(ratings[item_key]), list(range(M))))
3 user_inverse_mapper = dict(zip(list(range(N)), np.unique(ratings[user_k
4 item_inverse_mapper = dict(zip(list(range(M)), np.unique(ratings[item_k

```
In [8]: 1 def create_Y_from_ratings(
2     data, N, M, user_mapper, item_mapper, user_key="userId", item_key=""
3 ): # Function to create a dense utility matrix
4     Y = np.zeros((N, M))
5     Y.fill(np.nan)
6     for index, val in data.iterrows():
7         n = user_mapper[val[user_key]]
8         m = item_mapper[val[item_key]]
9         Y[n, m] = val["rating"]
10
11     return Y
```

Utility matrix for the example problem

- Rows represent users.
- Columns represent items (jokes in our case).
- Each cell gives the rating given by the user to the corresponding joke.
- Users are features for jokes and jokes are features for users.
- We want to predict the missing entries.

```
In [9]: 1 Y_mat = create_Y_from_ratings(ratings, N, M, user_mapper, item_mapper)
2 Y_mat.shape
```

Out[9]: (3635, 140)

```
In [10]: 1 pd.DataFrame(Y_mat)
```

	0	1	2	3	4	5	6	7	8	9	...	130	131	132
0	0.219	-9.281	-9.281	-6.781	0.875	-9.656	-9.031	-7.469	-8.719	-9.156	...	NaN	NaN	NaN
1	-9.688	9.938	9.531	9.938	0.406	3.719	9.656	-2.688	-9.562	-9.125	...	NaN	NaN	NaN
2	-9.844	-9.844	-7.219	-2.031	-9.938	-9.969	-9.875	-9.812	-9.781	-6.844	...	NaN	NaN	NaN
3	-5.812	-4.500	-4.906	NaN	...	NaN	NaN	NaN						
4	6.906	4.750	-5.906	-0.406	-4.031	3.875	6.219	5.656	6.094	5.406	...	NaN	NaN	NaN
...
3630	NaN	-9.812	-0.062	NaN	...	NaN	NaN	NaN						
3631	NaN	-9.844	7.531	-9.719	-9.344	3.875	9.812	8.938	8.375	NaN	...	NaN	NaN	NaN
3632	NaN	-1.906	3.969	-2.312	-0.344	-8.844	4.188	NaN	NaN	NaN	...	NaN	NaN	NaN
3633	NaN	-8.875	-9.156	-9.156	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
3634	NaN	-6.312	1.281	-3.531	2.125	-5.812	5.562	-6.062	0.125	NaN	...	NaN	NaN	4.1

3635 rows × 140 columns

In [11]:

```
1 print(np.sum(np.isfinite(Y_mat), axis=0).shape)
2 np.sum(np.isfinite(Y_mat), axis=0)
```

(140,)

Out[11]: array([661, 3625, 3542, 3441, 3371, 3325, 3286, 3184, 3142, 557, 951, 841, 837, 817, 849, 869, 368, 868, 1054, 825, 230, 1314, 811, 860, 1301, 1166, 836, 846, 854, 836, 837, 859, 166, 818, 839, 863, 909, 870, 944, 1246, 166, 167, 1368, 985, 829, 894, 826, 820, 825, 829, 222, 1063, 886, 821, 886, 1115, 824, 999, 1086, 852, 819, 1219, 167, 818, 828, 1079, 842, 853, 817, 167, 858, 844, 861, 834, 818, 834, 935, 856, 1481, 823, 961, 906, 1013, 853, 866, 876, 905, 857, 850, 168, 821, 846, 837, 959, 1149, 1141, 895, 1036, 865, 864, 923, 887, 860, 1364, 879, 166, 1080, 905, 1216, 890, 942, 859, 819, 813, 881, 966, 1321, 865, 1113, 879, 838, 1066, 855, 970, 883, 836, 864, 1098, 881, 830, 812, 876, 935, 862, 910, 823, 872, 967, 881, 973])

In [12]:

```
1 print(np.sum(np.isfinite(Y_mat), axis=1).shape)
2 np.sum(np.isfinite(Y_mat), axis=1)
```

(3635,)

Out[12]: array([62, 34, 18, ..., 6, 3, 11])

Baseline Approaches

- Recall that our goal is to predict missing entries in the utility matrix.

In [13]: 1 pd.DataFrame(Y_mat)

Out[13]:

	0	1	2	3	4	5	6	7	8	9	...	130	131	140
0	0.219	-9.281	-9.281	-6.781	0.875	-9.656	-9.031	-7.469	-8.719	-9.156	...	NaN	NaN	NaN
1	-9.688	9.938	9.531	9.938	0.406	3.719	9.656	-2.688	-9.562	-9.125	...	NaN	NaN	NaN
2	-9.844	-9.844	-7.219	-2.031	-9.938	-9.969	-9.875	-9.812	-9.781	-6.844	...	NaN	NaN	NaN
3	-5.812	-4.500	-4.906	NaN	...	NaN	NaN	NaN						
4	6.906	4.750	-5.906	-0.406	-4.031	3.875	6.219	5.656	6.094	5.406	...	NaN	NaN	NaN
...
3630	NaN	-9.812	-0.062	NaN	...	NaN	NaN	NaN						
3631	NaN	-9.844	7.531	-9.719	-9.344	3.875	9.812	8.938	8.375	NaN	...	NaN	NaN	NaN
3632	NaN	-1.906	3.969	-2.312	-0.344	-8.844	4.188	NaN	NaN	NaN	...	NaN	NaN	NaN
3633	NaN	-8.875	-9.156	-9.156	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
3634	NaN	-6.312	1.281	-3.531	2.125	-5.812	5.562	-6.062	0.125	NaN	...	NaN	NaN	4.15

3635 rows × 140 columns

Evaluation

- Although there is no notion of "accurate" recommendations, we need a way to evaluate our predictions so that we'll be able to compare different methods.
- Although we are doing unsupervised learning, we'll split the data and evaluate our predictions as follows.

Data splitting

- We split the ratings into train and validation sets.
- It's easier to split the ratings data instead of splitting the utility matrix.
- Don't worry about `y`; we're not really going to use it.

In [14]: 1 X = ratings.copy()
2 y = ratings[user_key]
3 X_train, X_valid, y_train, y_valid = train_test_split(
4 X, y, test_size=0.2, random_state=42
5)
6
7 X_train.shape, X_valid.shape

Out[14]: ((113089, 3), (28273, 3))

Now we will create utility matrices for train and validation splits.

```
In [15]: 1 train_mat = create_Y_from_ratings(X_train, N, M, user_mapper, item_mapper)
          2 valid_mat = create_Y_from_ratings(X_valid, N, M, user_mapper, item_mapper)
```

```
In [16]: 1 train_mat.shape, valid_mat.shape
```

```
Out[16]: ((3635, 140), (3635, 140))
```

- `train_mat` has only ratings from the train set and `valid_mat` has only ratings from the valid set.
- During training we assume that we do not have access to some of the available ratings. We predict these ratings and evaluate them against ratings in the validation set.

Questions for you

- How do train and validation utility matrices differ?
- Why are utility matrices for train and validation sets are of the same shape?

Answer:

- The training matrix `train_mat` is of shape N by M but only has ratings from `x_train` and all other ratings missing.
- The validation matrix `valid_mat` is also of shape N by M but it only has ratings `x_valid` and all other ratings missing.
- They have the same shape because both have the same number of users and items; that's how we have constructed them.

Evaluation

- Now that we have train and validation sets, how do we evaluate our predictions?
- You can calculate the error between actual ratings and predicted ratings with metrics of your choice.
 - Most common ones are MSE or RMSE.

- The `error` function below calculates RMSE and `evaluate` function prints train and validation RMSE.

In [17]:

```

1 def error(X1, X2):
2     """
3     Returns the root mean squared error.
4     """
5     return np.sqrt(np.nanmean((X1 - X2) ** 2))
6
7
8 def evaluate(pred_X, train_X, valid_X, model_name="Global average"):
9     print("%s train RMSE: %0.2f" % (model_name, error(pred_X, train_X)))
10    print("%s valid RMSE: %0.2f" % (model_name, error(pred_X, valid_X)))

```

Baselines

Let's first try some simple approaches to predict missing entries.

1. Global average baseline
2. [k-Nearest Neighbours imputation \(<https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html>\)](https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html)

Global average baseline

- Let's examine RMSE of the global average baseline.
- In this baseline we predict everything as the global average rating

In [18]:

```

1 avg = np.nanmean(train_mat)
2 pred_g = np.zeros(train_mat.shape) + avg
3 pd.DataFrame(pred_g).head()

```

Out[18]:

	0	1	2	3	4	5	6	7	8	9	...
0	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	...
1	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	...
2	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	...
3	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	...
4	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	1.20741	...

5 rows × 140 columns

In [19]:

```
1 evaluate(pred_g, train_mat, valid_mat, model_name="Global average")
```

Global average train RMSE: 5.75
 Global average valid RMSE: 5.77

\$k\$-nearest neighbours imputation (<https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html>)

- Can we try \$k\$-nearest neighbours type imputation?
- Impute missing values using the mean value from \$k\$ nearest neighbours found in the training set.
- Calculate distances between examples using features where neither value is missing.

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	?	?	2	?	3
User 2	3	?	?	?	?
User 3	?	5	4	?	5
User 4	?	?	?	?	?
User 5	?	?	?	5	?
User 6	?	5	4	3	?

In [20]:

```

1 from sklearn.impute import KNNImputer
2
3 imputer = KNNImputer(n_neighbors=10)
4 train_mat_imp = imputer.fit_transform(train_mat)

```

Notice that rows are users, so we will find nearest users (users that give similar ratings when rating the same jokes).

In [21]:

1 pd.DataFrame(train_mat_imp)

Out[21]:

	0	1	2	3	4	5	6	7	8	9	...	11
0	-5.9406	-9.2810	-9.2810	-6.7810	0.8750	-9.6560	-9.0310	-7.4690	-8.7190	-9.1560	...	-4.53
1	2.3405	9.9380	9.5310	9.9380	0.4060	3.7190	9.6560	-2.6880	4.3438	-9.1250	...	2.24
2	-9.8440	-3.5750	-7.2190	-2.0310	-9.9380	-9.9690	-9.8750	-9.8120	-9.7810	-6.8440	...	-4.41
3	-5.8120	-2.4624	-4.9060	-2.7781	-0.0532	-3.8594	1.7031	-0.3687	1.8469	0.0593	...	-2.03
4	1.3157	4.7500	1.8658	-0.4060	1.7937	3.8750	6.2190	1.9220	6.0940	5.4060	...	-0.28
...
3630	-0.7750	-9.8120	-0.0620	-2.8218	-4.1470	-4.8281	2.2718	-2.8782	-1.0125	0.0688	...	-6.68
3631	2.5188	-5.0625	-0.4001	-9.7190	-9.3440	-1.6408	-4.1187	8.9380	8.3750	-0.9314	...	-4.03
3632	0.1749	-1.9060	3.9690	-1.3844	-0.3440	-8.8440	4.1880	-1.5564	5.0593	0.3343	...	-4.01
3633	-4.5937	-6.4376	-5.9563	-9.1560	-7.1437	-5.5844	2.2531	-0.9688	-2.8530	-0.6406	...	-4.69
3634	-0.0812	-6.3120	1.2810	-3.5310	2.1250	-5.8120	5.5620	0.2218	0.1250	-1.1874	...	-3.81

In [22]:

1 evaluate(train_mat_imp, train_mat, valid_mat, model_name="KNN imputer")

KNN imputer train RMSE: 0.00

KNN imputer valid RMSE: 4.79

Finding nearest neighbors (<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html>)

- We can look at nearest neighbors of a query item.
- In that cases, we want rows to be jokes, and users to be features for jokes. With our current data frame, this means finding nearest neighbors of column vectors.

In [23]: 1 pd.DataFrame(train_mat_imp)

	0	1	2	3	4	5	6	7	8	9	...
0	-5.9406	-9.2810	-9.2810	-6.7810	0.8750	-9.6560	-9.0310	-7.4690	-8.7190	-9.1560	...
1	2.3405	9.9380	9.5310	9.9380	0.4060	3.7190	9.6560	-2.6880	4.3438	-9.1250	...
2	-9.8440	-3.5750	-7.2190	-2.0310	-9.9380	-9.9690	-9.8750	-9.8120	-9.7810	-6.8440	...
3	-5.8120	-2.4624	-4.9060	-2.7781	-0.0532	-3.8594	1.7031	-0.3687	1.8469	0.0593	...
4	1.3157	4.7500	1.8658	-0.4060	1.7937	3.8750	6.2190	1.9220	6.0940	5.4060	...
...
3630	-0.7750	-9.8120	-0.0620	-2.8218	-4.1470	-4.8281	2.2718	-2.8782	-1.0125	0.0688	...
3631	2.5188	-5.0625	-0.4001	-9.7190	-9.3440	-1.6408	-4.1187	8.9380	8.3750	-0.9314	...
3632	0.1749	-1.9060	3.9690	-1.3844	-0.3440	-8.8440	4.1880	-1.5564	5.0593	0.3343	...
3633	-4.5937	-6.4376	-5.9563	-9.1560	-7.1437	-5.5844	2.2531	-0.9688	-2.8530	-0.6406	...
3634	-0.0812	-6.3120	1.2810	-3.5310	2.1250	-5.8120	5.5620	0.2218	0.1250	-1.1874	...

3635 rows × 140 columns

(Optional) \$k\$-nearest neighbours on a query joke

- Let's transpose the matrix.

In [24]: 1 item_user_mat = train_mat_imp.T

In [25]: 1 jokes_df = pd.read_csv("../data/jester_items.csv")
2 jokes_df.head()

	jokeId	jokeText
0	1	A man visits the doctor. The doctor says "I have bad news for you. You have\ncancer and Alzheimer's disease". \nThe man replies "Well, thank God I don't have cancer!"\n
1	2	This couple had an excellent relationship going until one day he came home\nfrom work to find his girlfriend packing. He asked her why she was leaving him\nand she told him that she had heard awful things about him. \n\n"What could they possibly have said to make you move out?" \n\nThey told me that you were a pedophile." \n\nHe replied, "That's an awfully big word for a ten year old." \n
2	3	Q. What's 200 feet long and has 4 teeth? \n\nA. The front row at a Willie Nelson Concert.\n
3	4	Q. What's the difference between a man and a toilet? \n\nA. A toilet doesn't follow you around after you use it.\n
4	5	Q.\tWhat's O. J. Simpson's Internet address? \nA.\tSlash, slash, backslash, slash, escape.\n

In [26]: 1 id_joke_map = dict(zip(jokes_df.jokeId, jokes_df.jokeText))

In [27]:

```

1 from sklearn.neighbors import NearestNeighbors
2
3
4 def get_topk_recommendations(X, query_ind=0, metric="cosine", k=5):
5     query_idx = item_inverse_mapper[query_ind]
6     model = NearestNeighbors(n_neighbors=k, metric="cosine")
7     model.fit(X)
8     neigh_idx = model.kneighbors([X[query_idx]], k, return_distance=False)
9     neigh_idx = np.delete(neigh_idx, np.where(query_idx == query_idx))
10    recs = [id_joke_map[item_inverse_mapper[i]] for i in neigh_idx]
11    print("Query joke: ", id_joke_map[query_idx])
12
13    return pd.DataFrame(data=recs, columns=["top recommendations"])
14
15
16 get_topk_recommendations(item_user_mat, query_ind=8, metric="cosine", k=5)

```

Query joke: Q: If a person who speaks three languages is called "tri-lingual," and a person who speaks two languages is called "bi-lingual," what do you call a person who only speaks one language?

A: American!

Out[27]:

top recommendations

0	Q: What is the difference between George Washington, Richard Nixon, and Bill Clinton? A: Washington couldn't tell a lie, Nixon couldn't tell the truth, and Clinton doesn't know the difference. A man in a hot air balloon realized he was lost. He reduced altitude and spotted a woman below. He descended a bit more and shouted, "Excuse me, can you help me? I promised a friend I would meet him an hour ago, but I don't know where I am." The woman below replied, "You are in a hot air balloon hovering approximately 30 feet above the ground. You are between 40 and 41 degrees north latitude and between 59 and 60 degrees west longitude." "You must be an engineer," said the balloonist. "I am," replied the woman. "How did you know?" "Well," answered the balloonist, "everything you told me is technically correct, but I have no idea what to make of your information, and the fact is, I am still lost. Frankly, you've not been much help so far." The woman below responded, "You must be in management." "I am," replied the balloonist, "but how did you know?" "Well," said the woman, "you don't know where you are or where you are going. You have risen to where you are due to a large quantity of hot air. You made a promise that you have no idea how to keep, and you expect people beneath you to solve your problems. The fact is, you are in exactly the same position you were in before we met, but now, somehow, it's my fault!"
1	If pro- is the opposite of con- then congress must be the opposite of progress.
2	Arnold Schwarzenegger and Sylvester Stallone are making a movie about the lives of the great composers. Stallone says "I want to be Mozart." Schwarzenegger says: "In that case... I'll be Bach."

Question

- Instead of imputation, what would be the consequences if we replace `nan` with zeros so that we can calculate distances between vectors?

Answer

It's not a good idea to replace ratings with 0, because 0 can be an actual rating value in our case.

What to do with predictions?

- Once you have predictions, you can sort them based on ratings and recommend items with highest ratings.

Break (5 min)



Collaborative filtering

Collaborative filtering

- One of the most popular approach for recommendation systems.
- Approach used by the winning entry (and most of the entries) in the Netflix competition.
- An unsupervised approach
 - Only uses the user-item interactions given in the ratings matrix.
- Intuition**
 - We may have similar users and similar items which can help us predict missing entries.

Problem

- Given a utility matrix with many missing entries, how can we predict missing ratings?

\$\$ \begin{bmatrix} & ? & \checkmark & ? & \checkmark \\ ? & & \checkmark & ? & \checkmark \\ \checkmark & ? & & \checkmark & ? \\ ? & \checkmark & ? & & \checkmark \\ \checkmark & ? & ? & \checkmark & ? \end{bmatrix} \$\$

Note: rating prediction \neq Classification or regression

Classification or regression

- We have X and targets for some rows in X .
- We want to predict the last column (target column).

\$\$ \begin{bmatrix} \checkmark & \checkmark & \checkmark & \checkmark & \checkmark \\ \checkmark & \checkmark & \checkmark & \checkmark & \checkmark \\ \checkmark & \checkmark & \checkmark & \checkmark & \checkmark \\ \checkmark & ? & \checkmark & \checkmark & \checkmark \\ \checkmark & \checkmark & \checkmark & \checkmark & ? \end{bmatrix} \$\$

Rating prediction

- Ratings data has many missing values in the utility matrix. There is no special target column.
We want to predict the missing entries in the matrix.
- Since our goal is to **predict** ratings, usually the utility matrix is referred to as Y matrix.

\$\$ \begin{bmatrix} & ? & \checkmark & ? & \checkmark \\ ? & & \checkmark & ? & \checkmark \\ \checkmark & ? & & \checkmark & ? \\ ? & \checkmark & ? & & \checkmark \end{bmatrix} \$\$

- We don't have sufficient background to understand how collaborative filtering works under-the-hood.
- Let's look at an example to understand this at a high level.

```
In [28]: 1 toy_ratings = pd.read_csv("../data/toy-movie-ratings.csv")
          2 toy_ratings
```

Out[28]:

	user_id	movie_id	rating
0	Sam	Lion King	5
1	Sam	Toy Story	4
2	Sam	The Little Mermaid	5
3	Sam	Bambi	5
4	Sam	The Social Dilemma	1
5	Eva	Toy Story	1
6	Eva	The Social Dilemma	5
7	Eva	Man on Wire	5
8	Pat	The Little Mermaid	4
9	Pat	Lion King	5
10	Pat	Bambi	5
11	Jim	The Social Dilemma	5
12	Jim	Malcolm x	4
13	Jim	Man on Wire	5

```
In [29]: 1 N_toy = len(np.unique(toy_ratings["user_id"]))
          2 M_toy = len(np.unique(toy_ratings["movie_id"]))
          3 print("Number of users (N) : %d" % N_toy)
          4 print("Number of movies (M) : %d" % M_toy)
```

Number of users (N) : 4
 Number of movies (M) : 7

```
In [30]: 1 user_mapper_toy = dict(zip(np.unique(toy_ratings["user_id"]), list(range(N_toy)))
          2 item_mapper_toy = dict(zip(np.unique(toy_ratings["movie_id"]), list(range(M_toy))))
          3 user_inverse_mapper_toy = dict(
          4     zip(list(range(N_toy)), np.unique(toy_ratings["user_id"])))
          5 )
          6 item_inverse_mapper_toy = dict(
          7     zip(list(range(M_toy)), np.unique(toy_ratings["movie_id"])))
          8 )
```

```
In [31]: 1 Y_toy = create_Y_from_ratings(
2     toy_ratings, N_toy, M_toy, user_mapper_toy, item_mapper_toy, user_k
3 )
4 utility_mat_toy = pd.DataFrame(
5     Y_toy, columns=item_mapper_toy.keys(), index=user_mapper_toy.keys()
6 )
7 utility_mat_toy
```

Out[31]:

	Bambi	Lion King	Malcolm x	Man on Wire	The Little Mermaid	The Social Dilemma	Toy Story
Eva	NaN	NaN	NaN	5.0	NaN	5.0	1.0
Jim	NaN	NaN	4.0	5.0	NaN	5.0	NaN
Pat	5.0	5.0	NaN	NaN	4.0	NaN	NaN
Sam	5.0	5.0	NaN	NaN	5.0	1.0	4.0

- In this toy example, we see clear groups of movies and users.
 - For movies: Children movies and documentaries
 - For users: Children movie lovers and documentary lovers
- There are some unsupervised models which identify such latent features.
- I'll show you how to use a package which implements this popular algorithm for collaborative filtering.

Rating prediction using the surprise package

- We'll be using a package called [Surprise](https://surprise.readthedocs.io/en/stable/index.html) (<https://surprise.readthedocs.io/en/stable/index.html>).
- The collaborative filtering algorithm we use in this package is called SVD .

```
pip install scikit-surprise
```

Let's try it out on our Jester dataset utility matrix.

```
In [32]: 1 import surprise
2 from surprise import SVD, Dataset, Reader, accuracy
```

```
In [33]: 1 reader = Reader()
2 data = Dataset.load_from_df(ratings, reader) # Load the data
3
4 # I'm being sloppy here. Probably there is a way to create validset fro
5 trainset, validset = surprise.model_selection.train_test_split(
6     data, test_size=0.2, random_state=42
7 ) # Split the data
```

In [34]:

```

1 k = 10
2 algo = SVD(n_factors=k, random_state=42)
3 algo.fit(trainset)
4 svd_preds = algo.test(validset)
5 accuracy.rmse(svd_preds, verbose=True)

```

RMSE: 5.2893

Out[34]: 5.28926338380112

- No big improvement over the global baseline (RMSE=5.77).
- Probably because we are only considering a sample.

Cross-validation for recommender systems

- We can also carry out cross-validation and grid search with this package.
- Let's look at an example of cross-validation.

In [35]:

```

1 from surprise.model_selection import cross_validate
2
3 pd.DataFrame(cross_validate(algo, data, measures=["RMSE", "MAE"], cv=5),

```

Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	5.2756	5.2846	5.2603	5.2869	5.3155	5.2846	0.0181
MAE (testset)	4.1977	4.1922	4.1733	4.2036	4.2348	4.2003	0.0200
Fit time	0.23	0.23	0.22	0.22	0.22	0.22	0.01
Test time	0.08	0.15	0.08	0.08	0.15	0.11	0.04

Out[35]:

	test_rmse	test_mae	fit_time	test_time
0	5.275590	4.197694	0.231349	0.084050
1	5.284565	4.192235	0.228375	0.152414
2	5.260277	4.173313	0.217773	0.080490
3	5.286925	4.203588	0.216705	0.075303
4	5.315549	4.234839	0.215403	0.154287

Content-based filtering

What is content-based filtering?

- Supervised machine learning approach
- In collaborative filtering we assumed that we only have ratings data.
- Usually there is some information on items and users available.
- Examples
 - Netflix can describe movies as action, romance, comedy, documentaries.
 - Amazon could describe books according to topics: math, languages, history.
 - Tinder could describe people according to age, location, employment.
- Can we use this information to predict ratings in the utility matrix?
 - Yes!

Item and user features

- In collaborative filtering we assumed that we only have ratings data.
- Usually there is some information available on items and users.
- Examples
 - Netflix can describe movies as action, romance, comedy, documentaries.
 - Amazon could describe books according to topics: math, languages, history.
 - Tinder could describe people according to age, location, employment.
- Can we use this information to predict ratings in the utility matrix?
 - Yes!

Toy example: Movie recommendation

- Let's consider movie recommendation problem with the following toy data.

Ratings data

```
In [36]: 1 toy_ratings = pd.read_csv("../data/toy_ratings.csv")
2 toy_ratings
```

Out[36]:

	user_id	movie_id	rating
0	Sam	Lion King	4
1	Sam	Jerry Maguire	4
2	Sam	Roman Holidays	5
3	Sam	Downfall	1
4	Eva	Titanic	2
5	Eva	Jerry Maguire	1
6	Eva	Inception	4
7	Eva	Man on Wire	5
8	Eva	The Social Dilemma	5
9	Pat	Titanic	3
10	Pat	Lion King	4
11	Pat	Bambi	4
12	Pat	Cast Away	3
13	Pat	Jerry Maguire	5
14	Pat	Downfall	2
15	Pat	A Beautiful Mind	3
16	Jim	Titanic	2
17	Jim	Lion King	3
18	Jim	The Social Dilemma	5
19	Jim	Malcolm x	4
20	Jim	Man on Wire	5

```
In [37]: 1 N = len(np.unique(toy_ratings["user_id"]))
2 M = len(np.unique(toy_ratings["movie_id"]))
3 print("Number of users (N) : %d" % N)
4 print("Number of movies (M) : %d" % M)
```

Number of users (N) : 4
 Number of movies (M) : 12

```
In [38]: 1 user_key = "user_id"
2 item_key = "movie_id"
```

In [39]:

```

1 user_mapper = dict(zip(np.unique(toy_ratings[user_key]), list(range(N)))
2 item_mapper = dict(zip(np.unique(toy_ratings[item_key]), list(range(M))
3 user_inverse_mapper = dict(zip(list(range(N)), np.unique(toy_ratings[us
4 item_inverse_mapper = dict(zip(list(range(M)), np.unique(toy_ratings[it

```

Utility matrix

Let's create a dense utility matrix for our toy dataset.

In [40]:

```

1 def create_Y_from_ratings(data, N, M):
2     Y = np.zeros((N, M))
3     Y.fill(np.nan)
4     for index, val in data.iterrows():
5         n = user_mapper[val[user_key]]
6         m = item_mapper[val[item_key]]
7         Y[n, m] = val["rating"]
8
9     return Y

```

Utility matrix

In [41]:

```

1 Y = create_Y_from_ratings(toy_ratings, N, M)
2 utility_mat = pd.DataFrame(Y, columns=item_mapper.keys(), index=user_ma
3 utility_mat

```

Out[41]:

	A	Beautiful Mind	Bambi	Cast Away	Downfall	Inception	Jerry Maguire	Lion King	Malcolm x	Man on Wire	Roman Holidays	Titanic	Social Dilemma
Eva	NaN	NaN	NaN	NaN	4.0	1.0	NaN	NaN	5.0	NaN	NaN	5	NaN
Jim	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.0	4.0	5.0	NaN	5	NaN
Pat	3.0	4.0	3.0	2.0	NaN	NaN	5.0	4.0	NaN	NaN	NaN	NaN	NaN
Sam	NaN	NaN	NaN	1.0	NaN	4.0	4.0	NaN	NaN	5.0	NaN	NaN	NaN

In [42]:

```
1 avg = np.nanmean(Y)
```

Goal: Predict missing entries in the utility matrix.

In [43]:

```

1 import surprise
2 from surprise import SVD, Dataset, Reader, accuracy

```

Let's predict ratings with collaborative filtering.

In [44]:

```

1 reader = Reader()
2 data = Dataset.load_from_df(toy_ratings, reader) # Load the data
3
4 trainset, validset = surprise.model_selection.train_test_split(
5     data, test_size=0.01, random_state=42
6 ) # Split the data

```

In [45]:

```

1 k = 2
2 algo = SVD(n_factors=k, random_state=42)
3 algo.fit(trainset)
4 preds = algo.test(trainset.build_testset())

```

In [46]:

```

1 from collections import defaultdict
2
3 rating_preds = defaultdict(list)
4 for uid, iid, true_r, est, _ in preds:
5     rating_preds[uid].append((iid, est))

```

In [47]:

```
1 rating_preds
```

Out[47]:

```
defaultdict(list,
{'Sam': [('Lion King', 3.5442874862516582),
 ('Jerry Maguire', 3.471958396420975),
 ('Downfall', 3.141157981632877),
 ('Roman Holidays', 3.6555436348053982)],
 'Jim': [('Lion King', 3.6494404051925047),
 ('The Social Dilemma', 3.8739407581035588),
 ('Titanic', 3.295718235231984),
 ('Man on Wire', 3.8839492577938532),
 ('Malcolm x', 3.6435176323135128)],
 'Pat': [('A Beautiful Mind', 3.4463313322323263),
 ('Bambi', 3.540418795140043),
 ('Jerry Maguire', 3.4582870107738803),
 ('Titanic', 3.1872411557123517),
 ('Cast Away', 3.4442142132704827),
 ('Lion King', 3.5286392016604875),
 ('Downfall', 3.133747883605952)],
 'Eva': [('The Social Dilemma', 3.6665140635371194),
 ('Jerry Maguire', 3.3423360343482957),
 ('Titanic', 3.113324069881786),
 ('Man on Wire', 3.685575559931666)]})
```

Movie features

- Suppose we also have movie features.

In [48]:

```
1 movie_feats_df = pd.read_csv("../data/toy_movie_feats.csv", index_col=0)
2 movie_feats_df
```

Out[48]:

	Action	Romance	Drama	Comedy	Children	Documentary
A Beautiful Mind	0	1	1	0	0	0
Bambi	0	0	1	0	1	0
Cast Away	0	1	1	0	0	0
Downfall	0	0	0	0	0	1
Inception	1	0	1	0	0	0
Jerry Maguire	0	1	1	1	0	0
Lion King	0	0	1	0	1	0
Malcolm x	0	0	0	0	0	1
Man on Wire	0	0	0	0	0	1
Roman Holidays	0	1	1	1	0	0
The Social Dilemma	0	0	0	0	0	1
Titanic	0	1	1	0	0	0

In [49]:

```
1 Z = movie_feats_df.to_numpy()
2 Z.shape
```

Out[49]: (12, 6)

- How can we use these features to predict missing ratings?

Overall idea

- Using the ratings data and movie features, we'll build **profiles for different users**.
- Let's consider an example user **Pat**.

Pat's ratings

- We don't know anything about Pat but we know her ratings to movies.

```
In [50]: 1 utility_mat.loc["Pat"]
```

```
Out[50]: A Beautiful Mind      3.0
Bambi                         4.0
Cast Away                      3.0
Downfall                       2.0
Inception                      NaN
Jerry Maguire                  5.0
Lion King                      4.0
Malcolm x                     NaN
Man on Wire                    NaN
Roman Holidays                 NaN
The Social Dilemma             NaN
Titanic                        3.0
Name: Pat, dtype: float64
```

- We also know about movies and their features.
- If Pat gave a high rating to *Lion King*, it means that she liked the features of the movie.

```
In [51]: 1 movie_feats_df.loc["Lion King"]
```

```
Out[51]: Action          0
Romance         0
Drama           1
Comedy          0
Children        1
Documentary     0
Name: Lion King, dtype: int64
```

Supervised approach to rating prediction

- We treat ratings prediction problem as a set of regression problems.
- Given movie information, we create user profile for each user.
- Build regression model for each user and learn regression weights for each user.

- We build a profile for users based on
 - the movies they have watched
 - their rating for the movies
 - the features of the movies
- We train a personalized regression model for each user using this information.

Supervised approach to rating prediction

For each user i create a user profile as follows.

- Consider all movies rated by i and create x and y for the user:
 - Each row in x contains the movie features of movie j rated by i .
 - Each value in y is the corresponding rating given to the movie j by user i .

- Fit a regression model using `x` and `y`.
- Apply the model to predict ratings for new items!

Let's build user profiles

- Build `x` and `y` for all users.

```
In [52]: 1 from collections import defaultdict
2
3
4 def get_lr_data_per_user(ratings_df, d):
5     lr_y = defaultdict(list)
6     lr_X = defaultdict(list)
7     lr_items = defaultdict(list)
8
9     for index, val in ratings_df.iterrows():
10        n = user_mapper[val[user_key]]
11        m = item_mapper[val[item_key]]
12        lr_X[n].append(Z[m])
13        lr_y[n].append(val["rating"])
14        lr_items[n].append(m)
15
16    for n in lr_X:
17        lr_X[n] = np.array(lr_X[n])
18        lr_y[n] = np.array(lr_y[n])
19
20    return lr_X, lr_y, lr_items
```

```
In [53]: 1 d = movie_feats_df.shape[1]
2 X_train_usr, y_train_usr, rated_items = get_lr_data_per_user(toy_rating)
```

- What's going to be shape of each `x` and `y`?

Examine user profiles

- Let's examine some user profiles.

```
In [54]: 1 def get_user_profile(user_name):
2     X = X_train_usr[user_mapper[user_name]]
3     y = y_train_usr[user_mapper[user_name]]
4     items = rated_items[user_mapper[user_name]]
5     movie_names = [item_inverse_mapper[item] for item in items]
6     print("Profile for user: ", user_name)
7     profile_df = pd.DataFrame(X, columns=movie_feats_df.columns, index=)
8     profile_df["ratings"] = y
9     return profile_df
```

Pat's profile

In [55]: 1 get_user_profile("Pat")

Profile for user: Pat

Out[55]:

	Action	Romance	Drama	Comedy	Children	Documentary	ratings
Titanic	0	1	1	0	0	0	3
Lion King	0	0	1	0	1	0	4
Bambi	0	0	1	0	1	0	4
Cast Away	0	1	1	0	0	0	3
Jerry Maguire	0	1	1	1	0	0	5
Downfall	0	0	0	0	0	1	2
A Beautiful Mind	0	1	1	0	0	0	3

- Pat seems to like Children's movies and movies with Comedy.
- Seems like she's not so much into romantic movies.

Eva's profile

In [56]: 1 get_user_profile("Eva")

Profile for user: Eva

Out[56]:

	Action	Romance	Drama	Comedy	Children	Documentary	ratings
Titanic	0	1	1	0	0	0	2
Jerry Maguire	0	1	1	1	0	0	1
Inception	1	0	1	0	0	0	4
Man on Wire	0	0	0	0	0	1	5
The Social Dilemma	0	0	0	0	0	1	5

- Eva hasn't rated many movies. There are not many rows.
- Eva seems to like documentaries and action movies.
- Seems like she's not so much into romantic movies.

Regression models for users

```
In [57]: 1 from sklearn.linear_model import Ridge
2
3
4 def train_for_usr(user_name, model=Ridge()):
5     X = X_train_usr[user_mapper[user_name]]
6     y = y_train_usr[user_mapper[user_name]]
7     model.fit(X, y)
8     return model
9
10
11 def predict_for_usr(model, movie_names):
12     feat_vecs = movie_feats_df.loc[movie_names].values
13     preds = model.predict(feat_vecs)
14     return preds
```

Regression model for Pat

- What are the regression weights learned for Pat?

```
In [58]: 1 user_name = "Pat"
2 pat_model = train_for_usr(user_name)
3 col = "Coefficients for %s" % user_name
4 pd.DataFrame(pat_model.coef_, index=movie_feats_df.columns, columns=[co
```

Out[58]:

Coefficients for Pat	
Action	0.000000
Romance	-0.020833
Drama	0.437500
Comedy	0.854167
Children	0.458333
Documentary	-0.437500

Predictions for Pat

- How would Pat rate some movies she hasn't seen?

```
In [59]: 1 movies_to_pred = ["Roman Holidays", "Malcolm x"]
2 pred_df = movie_feats_df.loc[movies_to_pred]
3 pred_df
```

Out[59]:

	Action	Romance	Drama	Comedy	Children	Documentary
Roman Holidays	0	1	1	1	0	0
Malcolm x	0	0	0	0	0	1

```
In [60]: 1 user_name = "Pat"
2 preds = predict_for_usr(pat_model, movies_to_pred)
3 pred_df[user_name + "'s predicted ratings"] = preds
4 pred_df
```

Out[60]:

	Action	Romance	Drama	Comedy	Children	Documentary	Pat's predicted ratings
Roman Holidays	0	1	1	1	0	0	4.145833
Malcolm x	0	0	0	0	0	1	2.437500

Regression model for Eva

- What are the regression weights learned for Eva?

```
In [61]: 1 user_name = "Eva"
2 eva_model = train_for_usr(user_name)
3 col = "Coefficients for %s" % user_name
4 pd.DataFrame(eva_model.coef_, index=movie_feats_df.columns, columns=[co
```

Out[61]:

Coefficients for Eva	
Action	0.333333
Romance	-1.000000
Drama	-0.666667
Comedy	-0.666667
Children	0.000000
Documentary	0.666667

Predictions for Eva

- What are the predicted ratings for Eva for a list of movies?

In [62]:

```

1 user_name = "Eva"
2 preds = predict_for_usr(eva_model, movies_to_pred)
3 pred_df[user_name + "'s predicted ratings"] = preds
4 pred_df

```

Out[62]:

	Action	Romance	Drama	Comedy	Children	Documentary	Pat's predicted ratings	Eva's predicted ratings
Roman Holidays	0	1	1	1	0	0	4.145833	1.666667
Malcolm x	0	0	0	0	0	1	2.437500	4.666667

Completing the utility matrix with content-based filtering

Here is the original utility matrix.

In [63]:

```
1 utility_mat
```

Out[63]:

	A Beautiful Mind	Bambi	Cast Away	Downfall	Inception	Jerry Maguire	Lion King	Malcolm x	Man on Wire	Roman Holidays	The Social Dilemma
Eva	NaN	NaN	NaN	NaN	4.0	1.0	NaN	NaN	5.0	NaN	5
Jim	NaN	NaN	NaN	NaN	NaN	NaN	3.0	4.0	5.0	NaN	5
Pat	3.0	4.0	3.0	2.0	NaN	5.0	4.0	NaN	NaN	NaN	NaN
Sam	NaN	NaN	NaN	1.0	NaN	4.0	4.0	NaN	NaN	5.0	NaN

- Using predictions per user, we can fill in missing entries in the utility matrix.

In [64]:

```

1 from sklearn.linear_model import Ridge
2
3 models = dict()
4 pred_lin_reg = np.zeros((N, M))
5
6 for n in range(N):
7     models[n] = Ridge()
8     models[n].fit(X_train_usr[n], y_train_usr[n])
9     pred_lin_reg[n] = models[n].predict(Z)

```

In [65]: 1 pd.DataFrame(pred_lin_reg, columns=item_mapper.keys(), index=user_mappe

Out[65]:

	A Beautiful Mind	Bambi	Cast Away	Downfall	Inception	Jerry Maguire	Lion King	Malcolm x	Man on Wire	R Ho
Eva	2.333333	3.333333	2.333333	4.666667	3.666667	1.666667	3.333333	4.666667	4.666667	1.6
Jim	2.575000	3.075000	2.575000	4.450000	3.150000	2.575000	3.075000	4.450000	4.450000	2.5
Pat	3.291667	3.770833	3.291667	2.437500	3.312500	4.145833	3.770833	2.437500	2.437500	4.1
Sam	3.810811	3.675676	3.810811	1.783784	3.351351	4.270270	3.675676	1.783784	1.783784	4.2

More comments on content-based filtering

- The feature matrix for movies can contain different types of features.
 - Example: Plot of the movie (text features), actors (categorical features), year of the movie, budget and revenue of the movie (numerical features).
 - You'll apply our usual preprocessing techniques to these features.
- If you have enough data, you could also carry out hyperparameter tuning with cross-validation for each model.
- Finally, although we have been talking about linear models above, you can use any regression model of your choice.

Advantages of content-based filtering

- We don't need many users to provide ratings for an item.
- Each user is modeled separately, so you might be able to capture uniqueness of taste.
- Since you can obtain the features of the items, you can immediately recommend new items.
 - This would not have been possible with collaborative filtering.
- Recommendations are interpretable.
 - You can explain to the user why you are recommending an item because you have learned weights.

Disadvantages of content-based filtering

- Feature acquisition and feature engineering
 - What features should we use to explain the difference in ratings?
 - Obtaining those features for each item might be very expensive.
- Less diversity: hardly recommend an item outside the user's profile.
- Cold start: When a new user shows up, you don't have any information about them.

Hybrid filtering

- Combining advantages of collaborative filtering and content-based filtering

Final comments and summary

Formulating the problem of recommender systems

- We are given ratings data.
- We use this data to create **utility matrix** which encodes interactions between users and items.
- The utility matrix has many missing entries.
- We defined recommendation systems problem as **matrix completion problem**.

What did we cover?

- There is a big world of recommendation systems out there. We talked about some basic traditional approaches to recommender systems.
 - collaborative filtering
 - content-based filtering

If you want to know more advanced approaches to recommender systems, watch this 4-hour summer school tutorial by Xavier Amatriain, Research/Engineering Director @ Netflix.

- [Part1 \(<https://www.youtube.com/watch?v=bLhq63ygoU8>\)](https://www.youtube.com/watch?v=bLhq63ygoU8)
- [Part2 \(<https://www.youtube.com/watch?v=mRToFXINBpQ>\)](https://www.youtube.com/watch?v=mRToFXINBpQ)

Evaluation

- We split the data similar to supervised systems.
- We evaluate recommendation systems using traditional regression metrics such as MSE or RMSE.
- But real evaluation of recommender system can be very tricky because there is no ground truth.
- We have been using RMSE due to the lack of a better measure.
- What we actually want to measure is the interest that our user has in the recommended items.

Beyond error rate in recommendation systems

- If a system gives the best RMSE it doesn't necessarily mean that it's going to give best recommendations.
- In recommendation systems we do not have ground truth.

- Just training your model and evaluating it offline is not ideal.
- Other aspects such as simplicity, interpretation, code maintainability are equally (if not more) important than best validation error.
- Winning system of Netflix Challenge was never adopted.
 - Big mess of ensembles was not really maintainable
- There are other considerations.

Other issues important in recommender systems

Are these good recommendations?

You are looking for water shoes and at the moment you are looking at [VIFUUR Water Sports Shoes](https://www.amazon.ca/VIFUUR-Barefoot-Quick-Dry-Blue-38-39/dp/B0753DL15Y) (<https://www.amazon.ca/VIFUUR-Barefoot-Quick-Dry-Blue-38-39/dp/B0753DL15Y>), are these good recommendations?



Roll over image to zoom in



Sponsored

- Polyester and spandex
- Rubber sole
- Recommended 1: CONVENIENCE - Smooth neck design prevents chafing when wearing our water shoe. It is convenient to wear and take off.
- Recommended 2: COMFORTABLE FIT -- Breathable and smooth fabrics with fine stretch on uppers. Like socks, flexible and comfortable.
- Recommended 3: RUBBER OUTSOLE & FOOT SAFETY -- Wearable and top-quality rubber sole, which protects your feet from being hurt by sharp objects.
- Recommended 4: OCCASION - Yoga Training, beach, swimming, pool, weight training, wake-boarding, sailing, boating, kayaking, windsurfing, cycling, jogging, walking, fishing, beach volleyball, gardening, lawn, car-washing and driving. Family outings!
- Tips: VARIOUS SIZE AVAILABLE -- fit different feet, little kids, big kids, men, women are available.



Flux New Verano Polarized Sunglasses for Men and Women UV400,Anti-Slip,Adjustable Nose Pad

4.2 421

\$39.99

Sponsored

Have a question?

Find answers in product info, Q&As, reviews

Type your question or keyword

Products related to this item

Page 1 of 19

Sponsored



bopika Water Socks
Barefoot Shoes Water
Sports Shoes Quick-Dry
Aqua Yoga Socks for ...
 2,389
\$18.99



bopika Water Socks
Barefoot Shoes Water
Sports Shoes Quick-Dry
Aqua Yoga Socks for ...
 12
\$18.99



bopika Water Socks
Barefoot Shoes Water
Sports Shoes Quick-Dry
Aqua Yoga Socks for ...
 2,389
\$18.99



bopika Water Socks
Barefoot Shoes Water
Sports Shoes Quick-Dry
Aqua Yoga Socks for ...
 37
\$18.99



VALTEK Water Shoes
Paleos Barefoot Yoga
Socks Beach Swim Aqua
Surf Outdoor Shoes fo...
 125
\$22.99



bopika Water Socks
Barefoot Shoes Water
Sports Shoes Quick-Dry
Aqua Yoga Socks for ...
 33
\$18.99



bopika Water Socks
Barefoot Shoes Water
Sports Shoes Quick-Dry
Aqua Yoga Socks for ...
 24
\$18.99



bopika Water Socks
Barefoot Shoes Water
Sports Shoes Quick-Dry
Aqua Yoga Socks for ...
 24
\$18.99

Now suppose you've recently bought VIFUUR Water Sports Shoes and rated them highly. Are these good recommendations now?

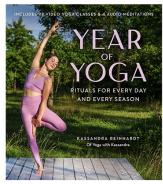
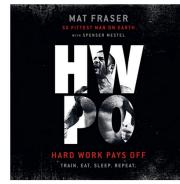
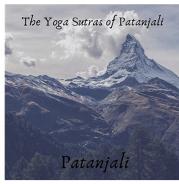
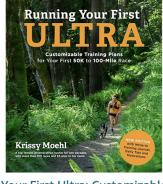
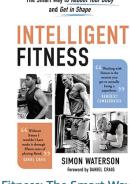
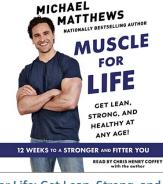
- Not really. Even though you really liked them you don't need them anymore. You want some non-Water Sports Shoes recommendations.
- **Diversity** is about how different the recommendations are.
 - Another example: Even if you really really like Star Wars, you might want non-Star-Wars suggestions.
- But be careful. We need a balance here.

Are these good recommendations?

Amazon Hot New Releases

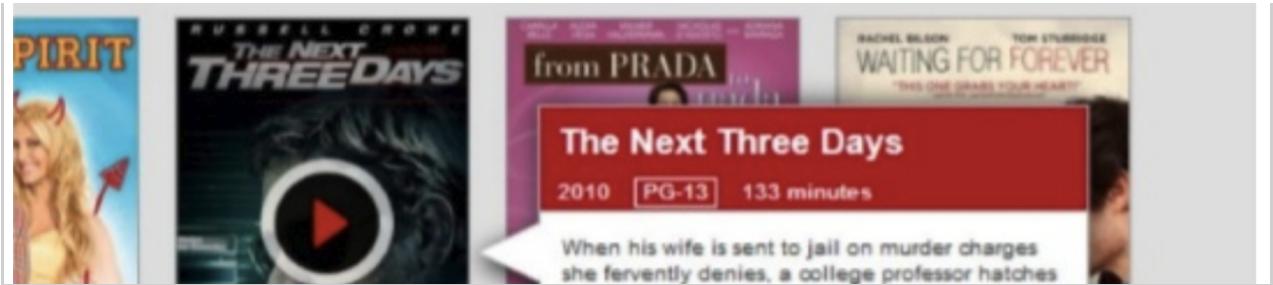
Our best-selling new and future releases. Updated hourly.

Hot New Releases in Exercise & Fitness

#1	#2	#3	#4
			
Year of Yoga: Rituals for Every Day and Every Season (Yoga with Kassandra, Yin Yoga,...) › Kassandra Reinhardt Paperback \$33.65	HWPO: Hard Work Pays Off Mat Fraser ★★★★★ 376 Audible Audiobook \$0.00 Free with Audible trial	75 HARD CHALLENGE BOOK: A TACTICAL GUIDE TO WINNING THE WAR WITH... 75 hard frisella Paperback \$16.99	The Yoga Sutras of Patanjali Patanjali ★★★★★ 470 Audible Audiobook \$0.00 Free with Audible trial
#5	#6	#7	#8
			
Running Your First Ultra: Customizable Training Plans for Your First 5K to 100-Mile... Moehl Krissy Paperback \$31.33	Intelligent Fitness: The Smart Way to Reboot... Simon Watson ★★★★★ 193 Kindle Edition \$9.99	Fat Girls Hiking: An Inclusive Guide to Getting Outdoors at Any Size or Ability Summer Michaud-Skog Paperback \$24.95	Muscle for Life: Get Lean, Strong, and Health... Michael Matthews ★★★★★ 207 Audible Audiobook \$0.00 Free with Audible trial

- Some of these books don't have many ratings but it might be a good idea to recommend "fresh" things.
- **Freshness:** people tend to get more excited about new/surprising things.

- But again you need a balance here. What would happen if you keep surprising the user all the time?
- There might be **trust** issues.
- Another aspect of trust is explaining your recommendation, i.e., telling the user why you made a recommendation. This gives the user an opportunity to understand why your recommendations could be interesting to them.



Persistence: how long should recommendations last?

- If you keep not clicking on a recommendation, should it remain a recommendation?

Social recommendation: what did your friends watch?

- Many recommenders are now connected to social networks.
- "Login using your Facebook account".
- Often, people like similar movies to their friends.
- If we get a new user, then recommendations are based on friend's preferences.

Types of data

- Explicit data: ratings, thumbs up, etc.
- Implicit data: collected from the users' behaviour (e.g., mouse clicks, purchases, time spent doing something)
- Trust implicit data that costs something, like time or even money.
 - this makes it harder to fraud

Some thoughts on recommendation systems

- Be mindful of the consequences of recommendation systems.
 - Recommendation systems can have terrible consequences.
- Companies such as Amazon, Netflix, Facebook, Google (YouTube), which extensively use recommendation systems, are profit-driven and so they design these systems to maximize user attention; their focus is not necessarily human well-being.
- There are tons of news and research articles on serious consequences of recommendation systems.

Some thoughts on recommendation systems

- Some weird stories which got media attention.

[How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did](#)
`(https://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/?sh=3171af136668)`
- More serious consequences are in political contexts.

- [Facebook Admits It Was Used to Incite Violence in Myanmar](https://www.nytimes.com/2018/11/06/technology/myanmar-facebook.html)
- [YouTube Extremism and the Long Tail](https://www.theatlantic.com/politics/archive/2018/03/youtube-extremism-and-the-long-tail/555250/)

My advice

- Ask hard and uncomfortable questions to yourself (and to your employer if possible) before implementing and deploying such systems.

Resources

- [Collaborative filtering for recommendation systems in Python, by N. Hug](https://www.youtube.com/watch?v=z0dx-YckFko)
- [An interesting talk: The paradox of choice](https://www.ted.com/talks/barry_schwartz_the_paradox_of_choice)
- [How Netflix's Recommendations System Works](https://help.netflix.com/en/node/100639)
- [Hands on Recommendation Systems with Python](https://learning.oreilly.com/library/view/hands-on-recommendation-systems/9781788993753/)

CPSC 330

Applied Machine Learning

Lecture 17: Introduction to natural language processing

UBC 2022-23

Instructor: Mathias Lécuyer

```
In [1]: 1 ## Imports
2
3 import os
4 import re
5 import string
6 import sys
7 import time
8 from collections import Counter, defaultdict
9
10 import IPython
11 import nltk
12 import numpy as np
13 import numpy.random as npr
14 import pandas as pd
15 from IPython.display import HTML
16 from ipywidgets import interactive
17 from nltk.corpus import stopwords
18 from nltk.tokenize import sent_tokenize, word_tokenize
19 from sklearn.feature_extraction.text import CountVectorizer
20 from sklearn.linear_model import LogisticRegression
21 from sklearn.pipeline import make_pipeline
```

Announcements

- Homework 7 due on the 22nd.
- Please take the instructor feedback survey:
https://canvas.ubc.ca/courses/30777/external_tools/6073
(https://canvas.ubc.ca/courses/30777/external_tools/6073).

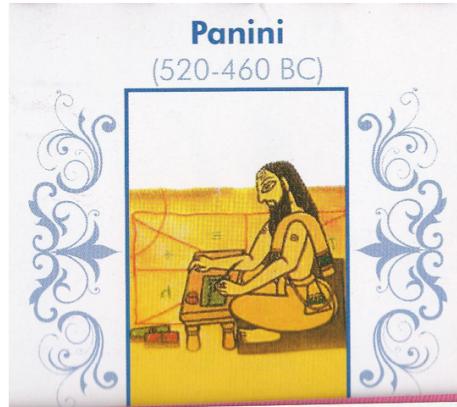
Learning objectives

- Broadly explain what is natural language processing (NLP).
- Name some common NLP applications.
- Explain the general idea of a vector space model.
- Explain the difference between different word representations: term-term co-occurrence matrix representation and Word2Vec representation.
- Describe the reasons and benefits of using pre-trained embeddings.
- Load and use pre-trained word embeddings to find word similarities and analogies.
- Demonstrate biases in embeddings and learn to watch out for such biases in pre-trained embeddings.
- Use word embeddings in text classification and document clustering using spaCy .
- Explain the general idea of topic modeling.
- Describe the input and output of topic modeling.
- Carry out basic text preprocessing using spaCy .

What is Natural Language Processing (NLP)?

- What should a search engine return when asked the following question?

Who is Panini?



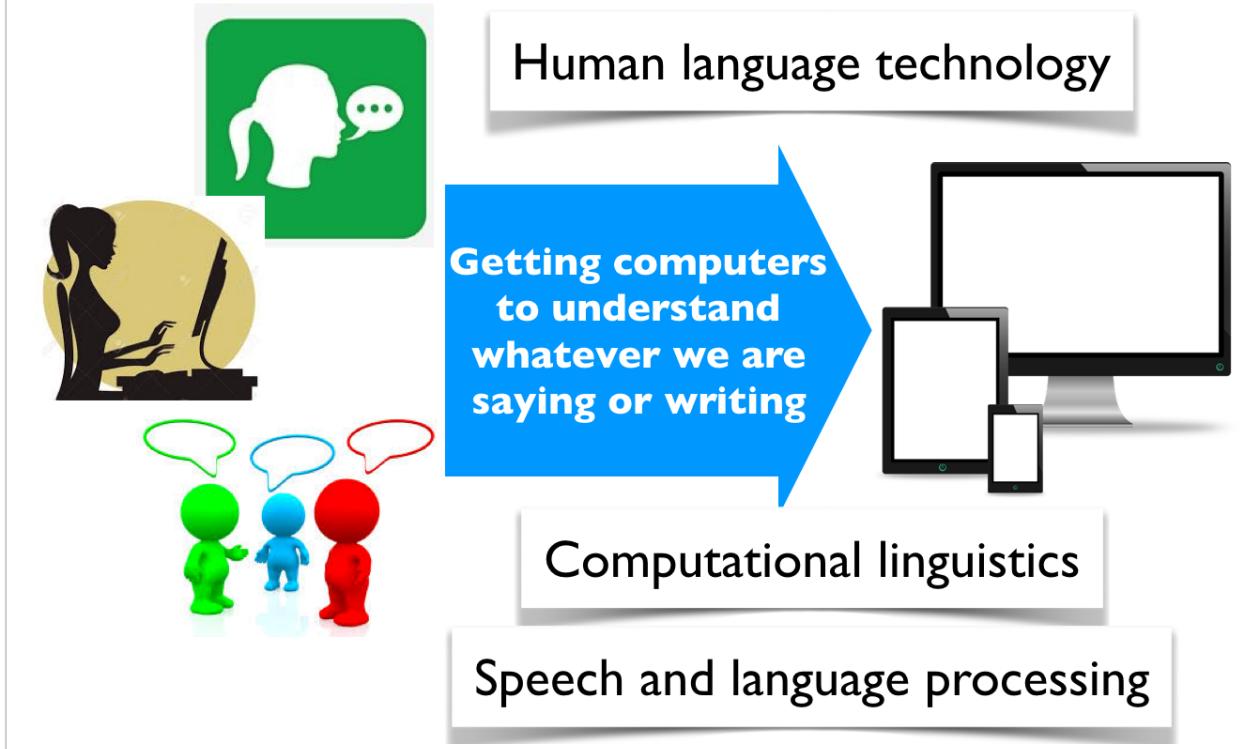
What is Natural Language Processing (NLP)?

How often do you search everyday?

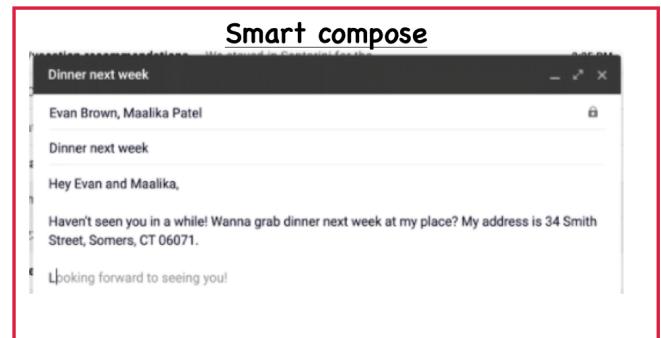
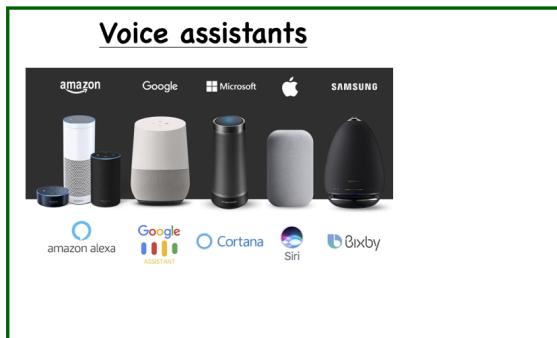
Google processes > **4.5 billion** queries per day!



What is Natural Language Processing (NLP)?



Everyday NLP applications



Translation

NLP in news

Often you'll see NLP in news. Some examples:

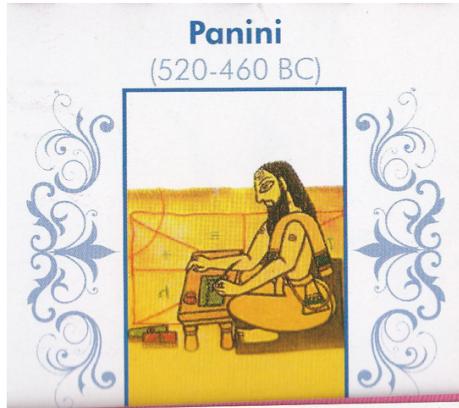
- [How suicide prevention is getting a boost from artificial intelligence](https://abcnews.go.com/GMA/Wellness/suicide-prevention-boost-artificial-intelligence-exclusive/story?id=76541481) (<https://abcnews.go.com/GMA/Wellness/suicide-prevention-boost-artificial-intelligence-exclusive/story?id=76541481>)
- [Meet GPT-3. It Has Learned to Code \(and Blog and Argue\).](https://www.nytimes.com/2020/11/24/science/artificial-intelligence-ai-gpt3.html) (<https://www.nytimes.com/2020/11/24/science/artificial-intelligence-ai-gpt3.html>)
- [Introducing ChatGPT](https://openai.com/blog/chatgpt) (<https://openai.com/blog/chatgpt>)
- [How Do You Know a Human Wrote This?](https://www.nytimes.com/2020/07/29/opinion/gpt-3-ai-automation.html) (<https://www.nytimes.com/2020/07/29/opinion/gpt-3-ai-automation.html>)
- ...

Why is NLP hard?

- Language is complex and subtle.
- Language is ambiguous at different levels.
- Language understanding involves common-sense knowledge and real-world reasoning.
- All the problems related to representation and reasoning in artificial intelligence arise in this domain.

Example: Lexical ambiguity

Who is Panini?



Example: Referential ambiguity

If the **baby** does not thrive on raw **milk**, boil **it**.



it = ?



[Ambiguous news headlines \(\[http://www.fun-with-words.com/ambiguous_headlines.html\]\(http://www.fun-with-words.com/ambiguous_headlines.html\)\)](http://www.fun-with-words.com/ambiguous_headlines.html)

PROSTITUTES APPEAL TO POPE

- **appeal to** means make a serious or urgent request or be attractive or interesting?

KICKING BABY CONSIDERED TO BE HEALTHY

- **kicking** is used as an adjective or a verb?

MILK DRINKERS ARE TURNING TO POWDER

- **turning** means becoming or take up?

Overall goal

- Give you a quick introduction to you of this important field in artificial intelligence which extensively used machine learning.

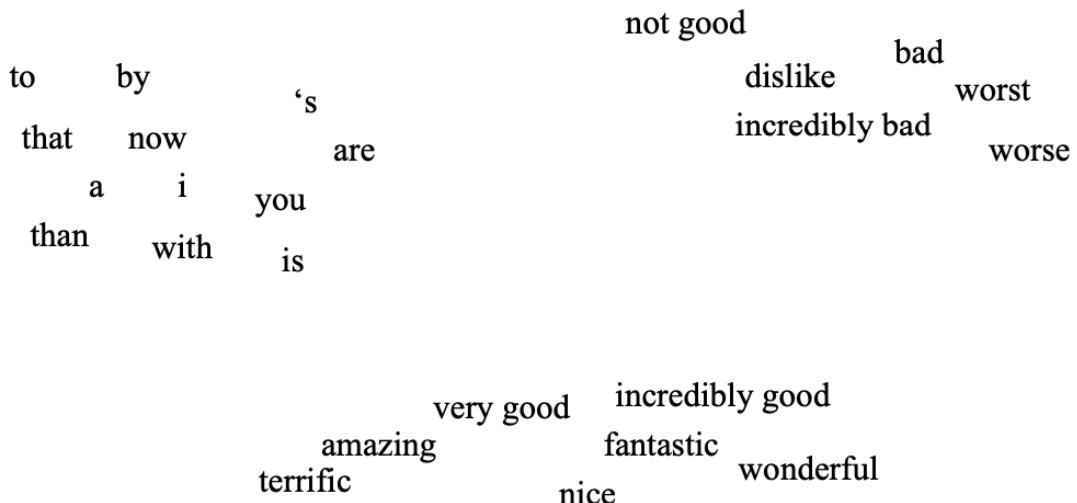


Today's plan

- Word embeddings
- Topic modeling
- Basic text preprocessing

Word Embeddings

- The idea is to represent word meaning so that similar words are close together.



Why do we care about word representation?

- So far we have been talking about sentence or document representation.
- Now we are going one step back and talking about word representation.
- Although word representation cannot be directly used in text classification tasks such as sentiment analysis using tradition ML models, it's good to know about word embeddings because they are so widely used.
- They are quite useful in more advanced machine learning models such as recurrent neural networks.

Word meaning

- A favourite topic of philosophers for centuries.
- An example from legal domain: [Are hockey gloves gloves or "articles of plastics"?](https://www.scc-csc.ca/case-dossier/info/sum-som-eng.aspx?cas=36258)

Canada (A.G.) v. Igloo Vikski Inc. was a tariff code case that made its way to the SCC (Supreme Court of Canada). The case disputed the definition of hockey gloves as either gloves or as "articles of plastics."



Word meaning: ML and NLP view

- Modeling word meaning that allows us to
 - draw useful inferences to solve meaning-related problems
 - find relationship between words,
 - E.g., which words are similar, which ones have positive or negative connotations

Word representations

How do we represent words?

- Suppose you are building a question answering system and you are given the following question and three candidate answers.
- What kind of relationship between words would we like our representation to capture in order to arrive at the correct answer?

Question: How tall is Machu Picchu?

Candidate 1: Machu Picchu is 13.164 degrees south of the equator.

Candidate 2: The official height of Machu Picchu is 2,430 m.

Candidate 3: Machu Picchu is 80 kilometres (50 miles) northwest of Cusco.

Need a representation that captures relationships between words.

- We will be looking at two such representations.
 1. Sparse representation with **term-term co-occurrence matrix**
 2. Dense representation with **Word2Vec**
- Both are based on two ideas: **distributional hypothesis** and **vector space model**.

Distributional hypothesis

You shall know a word by the company it keeps.

— Firth, 1957

If A and B have almost identical environments we say that they are synonyms.

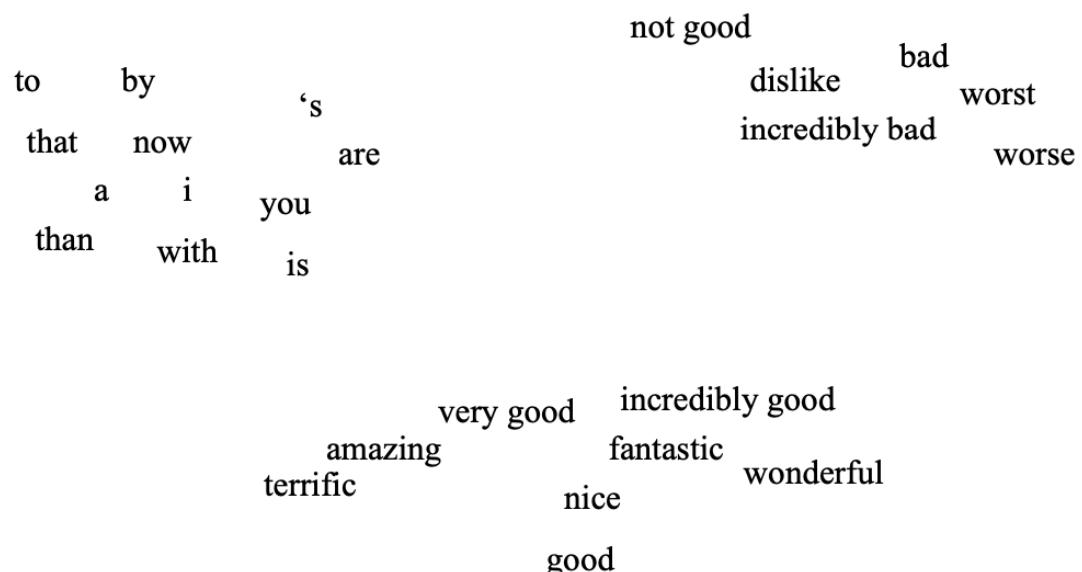
— Harris, 1954

Example:

- Her **child** loves to play in the playground.
- Her **kid** loves to play in the playground.

Vector space model

- Model the meaning of a word by placing it into a vector space.
- A standard way to represent meaning in NLP
- The idea is to create **embeddings of words** so that distances among words in the vector space indicate the relationship between them.



(Attribution: Jurafsky and Martin 3rd edition)

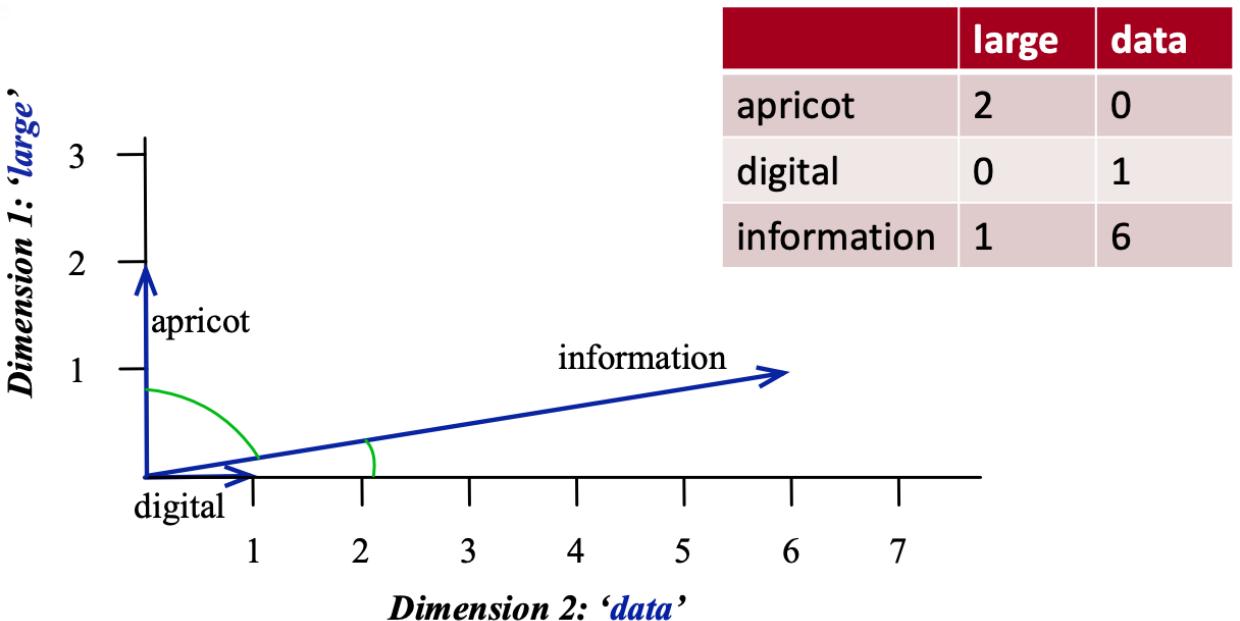
Term-term co-occurrence matrix

- So far we have been talking about documents and we created document-term co-occurrence matrix (e.g., bag-of-words representation of text).
- We can also do this with words. The idea is to go through a corpus of text, keeping a count of all of the words that appear in context of each word (within a window).
- An example:

	large	data
apricot	2	0
digital	0	1
information	1	6

(Credit: Jurafsky and Martin 3rd edition)

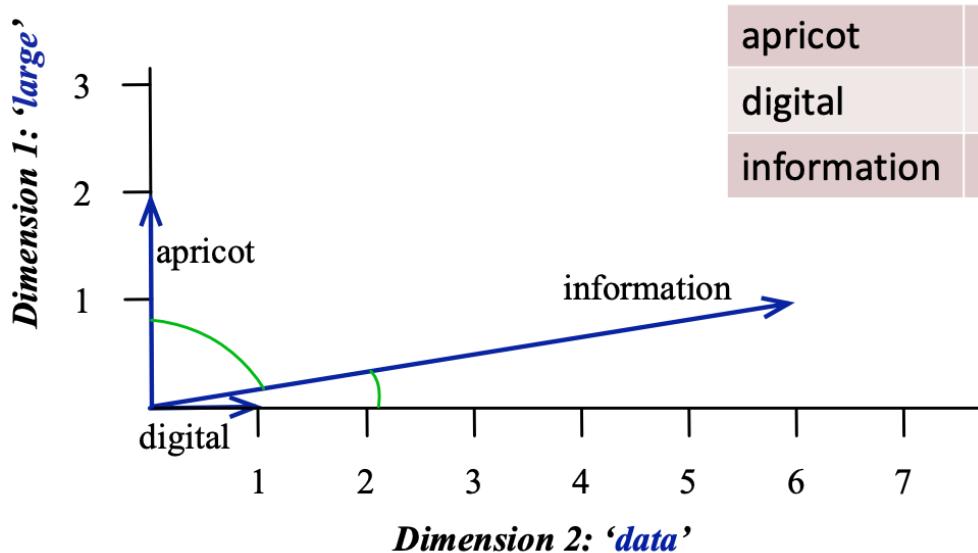
Visualizing word vectors and similarity



(Credit: Jurafsky and Martin 3rd edition)

- The similarity is calculated using dot products between word vectors.
 - Example: $\vec{\text{digital}} \cdot \vec{\text{information}} = 0 \times 1 + 1 \times 6 = 6$
 - Higher the dot product more similar the words.

Visualizing word vectors and similarity



(Credit: Jurafsky and Martin 3rd edition)

- The similarity is calculated using dot products between word vectors.
 - Example: $\vec{\text{digital}} \cdot \vec{\text{information}} = 0 \times 1 + 1 \times 6 = 6$
 - Higher the dot product more similar the words.
- We can also calculate a normalized version of dot products (related to the angle between word vectors).

$$\text{similarity}_{\text{cosine}}(w_1, w_2) = \frac{w_1 \cdot w_2}{\|w_1\|_2 \|w_2\|_2}$$

In [2]:

```

1 ### Let's build term-term co-occurrence matrix for our text.
2 sys.path.append("../code/.")
3
4 from comat import CooccurrenceMatrix
5 from preprocessing import MyPreprocessor
6
7 corpus = [
8     "How tall is Machu Picchu?",
9     "Machu Picchu is 13.164 degrees south of the equator.",
10    "The official height of Machu Picchu is 2,430 m.",
11    "Machu Picchu is 80 kilometres (50 miles) northwest of Cusco.",
12    "It is 80 kilometres (50 miles) northwest of Cusco, on the crest of"
13]
14 pp = MyPreprocessor()
15 pp_corpus = pp.preprocess_corpus(corpus)
16 cm = CooccurrenceMatrix(pp_corpus)
17 vocab, comat = cm.fit_transform()
18 words = [
19     key for key, value in sorted(vocab.items(), key=lambda item: (item[0]))
20]
21 df = pd.DataFrame(comat.todense(), columns=words, index=words, dtype=np
22 df.head()

```

Out[2]:

	tall	machu	picchu	13.164	degrees	south	equator	official	height	2,430	...	mean	se
tall	0	1	1	0	0	0	0	0	0	0	0	0	1
machu	1	0	5	1	1	0	0	1	1	2	...	0	1
picchu	1	5	0	1	1	1	0	1	1	2	...	0	1
13.164	0	1	1	0	1	1	1	0	0	0	0	0	1
degrees	0	1	1	1	0	1	1	0	0	0	0	0	1

5 rows × 32 columns

In [3]:

```

1 from sklearn.metrics.pairwise import cosine_similarity
2 import warnings
3 warnings.simplefilter(action='ignore', category=FutureWarning)
4
5 def similarity(word1, word2):
6     """
7         Returns similarity score between word1 and word2
8         Arguments
9         -----
10        word1 -- (str)
11            The first word
12        word2 -- (str)
13            The second word
14
15        Returns
16        -----
17        None. Prints the similarity score between word1 and word2.
18        """
19        vec1 = cm.get_word_vector(word1).todense().flatten()
20        vec2 = cm.get_word_vector(word2).todense().flatten()
21        v1 = np.squeeze(np.asarray(vec1))
22        v2 = np.squeeze(np.asarray(vec2))
23        print(
24            "The dot product between %s and %s is %0.2f and cosine similari
25            % (word1, word2, np.dot(v1, v2), cosine_similarity(vec1, vec2))
26        )
27
28
29 similarity("tall", "height")
30 similarity("tall", "official")
31
32 ### Not very reliable similarity scores because we used only 4 sentence

```

The dot product between tall and height is 2.00 and cosine similarity is 0.71

The dot product between tall and official is 2.00 and cosine similarity is 0.82

- We are able to capture some similarities between words now.
- That said similarities do not make much sense in the toy example above because we're using a tiny corpus.
- To find meaningful patterns of similarities between words, we need a large corpus.
- Let's try a bit larger corpus and check whether the similarities make sense.

In [4]:

```

1 import wikipedia
2 from nltk.tokenize import sent_tokenize, word_tokenize
3
4 corpus = []
5
6 queries = ["Machu Picchu", "human stature"]
7
8 for i in range(len(queries)):
9     sents = sent_tokenize(wikipedia.page(queries[i]).content)
10    corpus.extend(sents)
11 print("Number of sentences in the corpus: ", len(sents))

```

Number of sentences in the corpus: 269

In [5]:

```

1 pp = MyPreprocessor()
2 pp_corpus = pp.preprocess_corpus(corpus)
3 cm = CooccurrenceMatrix(pp_corpus)
4 vocab, comat = cm.fit_transform()
5 words = [
6     key for key, value in sorted(vocab.items(), key=lambda item: (item[1]))
7 ]
8 df = pd.DataFrame(comat.todense(), columns=words, index=words, dtype=np.int)
9 df.head()

```

Out[5]:

	machu	picchu	15th-century	inca	citadel	located	eastern	cordillera	southern	peru	...	cha
machu	0	81	1	5	1	2	1	0	1	2	...	
picchu	81	0	1	6	3	1	1	0	1	1	...	
15th-century	1	1	0	1	1	1	0	0	0	0	...	
inca	5	6	1	0	1	1	1	0	0	1	...	
citadel	1	3	1	1	0	1	1	1	0	0	...	

5 rows × 3019 columns

In [6]:

```

1 similarity("tall", "height")
2 similarity("tall", "official")

```

The dot product between tall and height is 72.00 and cosine similarity is 0.23

The dot product between tall and official is 0.00 and cosine similarity is 0.00

Sparse vs. dense word vectors

- Term-term co-occurrence matrices are long and sparse.
 - length $|V|$ is usually large (e.g., $> 50,000$)

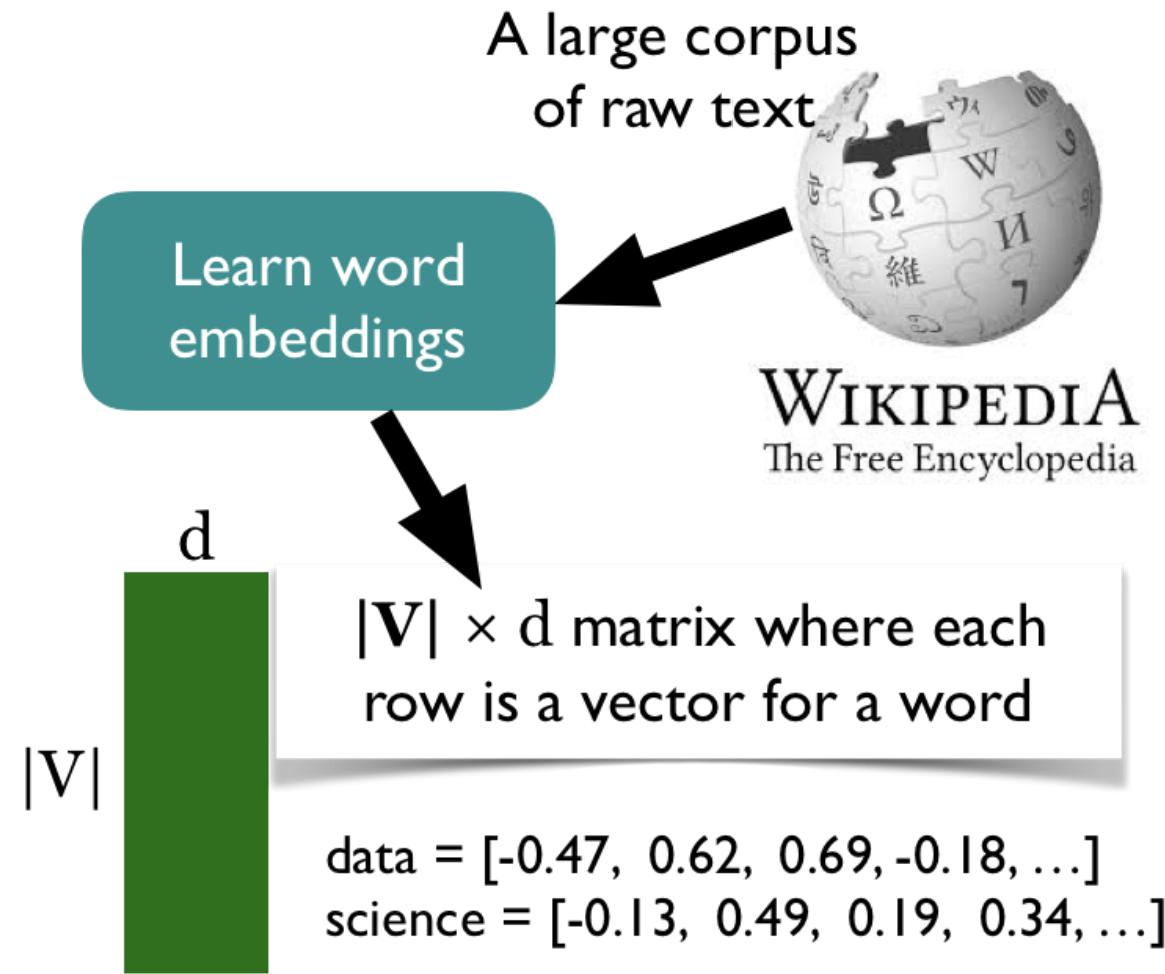
- most elements are zero
- OK because there are efficient ways to deal with sparse matrices.

Alternative

- Learn short (~100 to 1000 dimensions) and dense vectors.
- Short vectors may be easier to train with ML models (less weights to train).
- They may generalize better.
- In practice they work much better!

Word2Vec

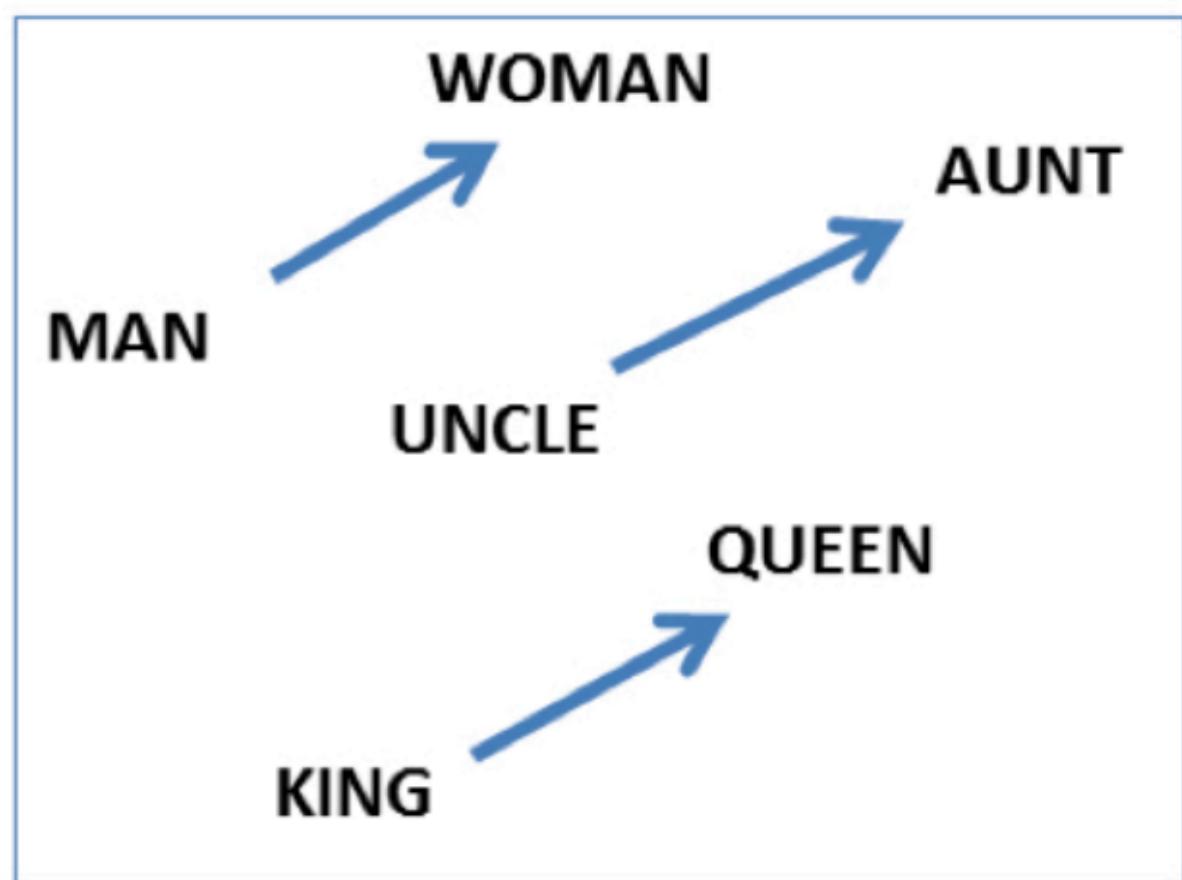
- A family of algorithms to create dense word embeddings



Success of Word2Vec

- Able to capture complex relationships between words.
- Example: What is the word that is similar to **WOMAN** in the same sense as **KING** is similar to **MAN**?
- Perform a simple algebraic operations with the vector representation of words.

$$\vec{X} = \vec{\text{KING}} - \vec{\text{MAN}} + \vec{\text{WOMAN}}$$
- Search in the vector space for the word closest to \vec{X} measured by cosine distance.



(Credit: Mikolov et al. 2013)

- We can create a dense representation with a library called `gensim`.

```
conda install -c anaconda gensim
```

In [7]:

```
1 from gensim.models import Word2Vec  
2  
3 sentences = [[ "cat", "say", "meow" ], [ "dog", "say", "woof" ]]  
4 model = Word2Vec(sentences, min_count=1)
```

Let's look at the word vector of the word *cat*.

In [8]: 1 model.wv["cat"]

```
Out[8]: array([-0.00713902,  0.00124103, -0.00717672, -0.00224462,  0.0037193 ,
   0.00583312,  0.00119818,  0.00210273, -0.00411039,  0.00722533,
  -0.00630704,  0.00464722, -0.00821997,  0.00203647, -0.00497705,
  -0.00424769, -0.00310898,  0.00565521,  0.0057984 , -0.00497465,
  0.00077333, -0.00849578,  0.00780981,  0.00925729, -0.00274233,
  0.00080022,  0.00074665,  0.00547788, -0.00860608,  0.00058446,
  0.00686942,  0.00223159,  0.00112468, -0.00932216,  0.00848237,
  -0.00626413, -0.00299237,  0.00349379, -0.00077263,  0.00141129,
  0.00178199, -0.0068289 , -0.00972481,  0.00904058,  0.00619805,
  -0.00691293,  0.00340348,  0.00020606,  0.00475375, -0.00711994,
  0.00402695,  0.00434743,  0.00995737, -0.00447374, -0.00138926,
  -0.00731732, -0.00969783, -0.00908026, -0.00102275, -0.00650329,
  0.00484973, -0.00616403,  0.00251919,  0.00073944, -0.00339215,
  -0.00097922,  0.00997913,  0.00914589, -0.00446183,  0.00908303,
  -0.00564176,  0.00593092, -0.00309722,  0.00343175,  0.00301723,
  0.00690046, -0.00237388,  0.00877504,  0.00758943, -0.00954765,
  -0.00800821, -0.0076379 ,  0.00292326, -0.00279472, -0.00692952,
  -0.00812826,  0.00830918,  0.00199049, -0.00932802, -0.00479272,
  0.00313674, -0.00471321,  0.00528084, -0.00423344,  0.0026418 ,
  -0.00804569,  0.00620989,  0.00481889,  0.00078719,  0.00301345],
  dtype=float32)
```

What's the most similar word to the word cat?

In [9]: 1 model.wv.most_similar("cat")

```
Out[9]: [('dog', 0.17018887400627136),
 ('woof', 0.004502999130636454),
 ('say', -0.027750356122851372),
 ('meow', -0.04461711645126343)]
```

This is good. But if you want good and meaningful representations of words you need to train models on a large corpus such as the whole Wikipedia, which is computationally intensive.

So instead of training our own models, we use the **pre-trained embeddings**. These are the word embeddings people have trained embeddings on huge corpora and made them available for us to use.

Let's try out Google news pre-trained word vectors.

In [10]: 1 # It'll take a while to run this when you try it out for the first time
2 import gensim.downloader as api
3
4 google_news_vectors = api.load("word2vec-google-news-300")

In [11]: 1 print("Size of vocabulary: ", len(google_news_vectors))

Size of vocabulary: 3000000

- google_news_vectors above has 300 dimensional word vectors for 3,000,000 unique words from Google news.

- What can we do with these word vectors?

Finding similar words

- Given word w , search in the vector space for the word closest to w as measured by cosine distance.

In [12]: 1 google_news_vectors.most_similar("UBC")

Out[12]: [('UVic', 0.788647472858429),
 ('SFU', 0.7588527202606201),
 ('Simon_Fraser', 0.7356573939323425),
 ('UFV', 0.6880434155464172),
 ('VIU', 0.6778583526611328),
 ('Kwantlen', 0.6771427989006042),
 ('UBCO', 0.6734487414360046),
 ('UPEI', 0.6731125116348267),
 ('UBC_Okanagan', 0.6709135174751282),
 ('Lakehead_University', 0.662250816822052)]

In [13]: 1 google_news_vectors.most_similar("information")

Out[13]: [('info', 0.7363681793212891),
 ('infomation', 0.680029571056366),
 ('infor_mation', 0.6733849048614502),
 ('informaiton', 0.6639008522033691),
 ('informa_tion', 0.6601257920265198),
 ('informationon', 0.633933424949646),
 ('informationabout', 0.6320980787277222),
 ('Information', 0.6186580657958984),
 ('informaion', 0.6093292236328125),
 ('details', 0.6063088178634644)]

If you want to extract all documents containing information or similar words, you could use this information.

Finding similarity scores between words

In [14]: 1 google_news_vectors.similarity("Canada", "hockey")

Out[14]: 0.27610135

In [15]: 1 google_news_vectors.similarity("China", "hockey")

Out[15]: 0.060384665

In [16]:

```

1 word_pairs = [
2     ("height", "tall"),
3     ("pineapple", "mango"),
4     ("pineapple", "juice"),
5     ("sun", "robot"),
6     ("GPU", "lion"),
7 ]
8 for pair in word_pairs:
9     print(
10         "The similarity between %s and %s is %0.3f"
11         % (pair[0], pair[1], google_news_vectors.similarity(pair[0], pa
12     )

```

The similarity between height and tall is 0.473

The similarity between pineapple and mango is 0.668

The similarity between pineapple and juice is 0.418

The similarity between sun and robot is 0.029

The similarity between GPU and lion is 0.002

In [17]:

```

1 def analogy(word1, word2, word3, model=google_news_vectors):
2     """
3         Returns analogy word using the given model.
4
5         Parameters
6         -----
7         word1 : (str)
8             word1 in the analogy relation
9         word2 : (str)
10            word2 in the analogy relation
11         word3 : (str)
12             word3 in the analogy relation
13         model :
14             word embedding model
15
16         Returns
17         -----
18             pd.DataFrame
19         """
20     print("%s : %s :: %s : ?" % (word1, word2, word3))
21     sim_words = model.most_similar(positive=[word3, word2], negative=[w
22     return pd.DataFrame(sim_words, columns=["Analogy word", "Score"])

```

```
In [18]: 1 analogy("man", "king", "woman")
```

man : king :: woman : ?

```
Out[18]:
```

	Analogy word	Score
--	--------------	-------

	Analogy word	Score
0	queen	0.711819
1	monarch	0.618967
2	princess	0.590243
3	crown_prince	0.549946
4	prince	0.537732
5	kings	0.523684
6	Queen_Consort	0.523595
7	queens	0.518113
8	sultan	0.509859
9	monarchy	0.508741

```
In [19]: 1 analogy("Montreal", "Canadiens", "Vancouver")
```

Montreal : Canadiens :: Vancouver : ?

```
Out[19]:
```

	Analogy word	Score
--	--------------	-------

	Analogy word	Score
0	Canucks	0.821327
1	Vancouver_Canucks	0.750401
2	Calgary_Flames	0.705471
3	Leafs	0.695783
4	Maple_Leafs	0.691617
5	Thrashers	0.687503
6	Avs	0.681716
7	Sabres	0.665307
8	Blackhawks	0.664625
9	Habs	0.661023

In [20]: 1 analogy("Toronto", "UofT", "Vancouver")

Toronto : UofT :: Vancouver : ?

Out[20]:

	Analogy word	Score
0	SFU	0.579245
1	UVic	0.576921
2	UBC	0.571431
3	Simon_Fraser	0.543464
4	Langara_College	0.541347
5	UVIC	0.520495
6	Grant_MacEwan	0.517273
7	UFV	0.514150
8	Ubyssey	0.510421
9	Kwantlen	0.503807

Examples of semantic and syntactic relationships

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

(Credit: Mikolov 2013)

Implicit biases and stereotypes in word embeddings

- Reflect gender stereotypes present in broader society.

- They may also amplify these stereotypes because of their widespread usage.
- See the paper [Man is to Computer Programmer as Woman is to ...](http://papers.nips.cc/paper/6228-man-is-to-computer-programmer-as-woman-is-to-homemaker-debiasing-word-embeddings.pdf) (<http://papers.nips.cc/paper/6228-man-is-to-computer-programmer-as-woman-is-to-homemaker-debiasing-word-embeddings.pdf>).

In [21]: 1 analogy("man", "computer_programmer", "woman")

man : computer_programmer :: woman : ?

Out[21]:

	Analogy word	Score
0	homemaker	0.562712
1	housewife	0.510505
2	graphic_designer	0.505180
3	schoolteacher	0.497949
4	businesswoman	0.493489
5	paralegal	0.492551
6	registered_nurse	0.490797
7	saleswoman	0.488163
8	electrical_engineer	0.479773
9	mechanical_engineer	0.475540



Most of the modern embeddings are de-biased.

(NLP)### Other pre-trained embeddings

A number of pre-trained word embeddings are available. The most popular ones are:

- [Word2Vec \(https://code.google.com/archive/p/word2vec/\)](https://code.google.com/archive/p/word2vec/)
 - trained on several corpora using the word2vec algorithm
- [wikipedia2vec \(https://github.com/davidsopp/wiki2vec\)](https://github.com/davidsopp/wiki2vec)
 - pretrained embeddings for 12 languages
- [GloVe \(https://nlp.stanford.edu/projects/glove/\)](https://nlp.stanford.edu/projects/glove/)
 - trained using [the GloVe algorithm \(https://nlp.stanford.edu/pubs/glove.pdf\)](https://nlp.stanford.edu/pubs/glove.pdf)
 - published by Stanford University
- [fastText pre-trained embeddings for 294 languages \(https://fasttext.cc/docs/en/pretrained-vectors.html\)](https://fasttext.cc/docs/en/pretrained-vectors.html)
 - trained using [the fastText algorithm \(http://aclweb.org/anthology/Q17-1010\)](http://aclweb.org/anthology/Q17-1010)
 - published by Facebook

Note that these pre-trained word vectors are of big size (in several gigabytes).

Here is a list of all pre-trained embeddings available with `gensim`.

In [22]:

```
1 import gensim.downloader
2
3 print(list(gensim.downloader.info()["models"].keys()))
```

['fasttext-wiki-news-subwords-300', 'conceptnet-numberbatch-17-06-300',
 'word2vec-ruscorpora-300', 'word2vec-google-news-300', 'glove-wiki-gigaword-50',
 'glove-wiki-gigaword-100', 'glove-wiki-gigaword-200', 'glove-wiki-gigaword-300',
 'glove-twitter-25', 'glove-twitter-50', 'glove-twitter-100', 'glove-twitter-200',
 '__testing_word2vec-matrix-synopsis']

Word vectors with spaCy

- spaCy also gives you access to word vectors with bigger models: `en_core_web_md` or `en_core_web_lr`
- spaCy's pre-trained embeddings are trained on [OntoNotes corpus \(https://catalog.ldc.upenn.edu/LDC2013T19\)](https://catalog.ldc.upenn.edu/LDC2013T19).
- This corpus has a collection of different styles of texts such as telephone conversations, newswire, newsgroups, broadcast news, broadcast conversation, weblogs, religious texts.
- Let's try it out.

In [23]:

```
1 import spacy
2
3 nlp = spacy.load("en_core_web_md")
4
5 doc = nlp("pineapple")
6 doc.vector
```

```
Out[23]: array([[ 6.5486e-01, -2.2584e+00,  6.2793e-02,  1.8801e+00,  2.0700e-01,
   -3.3299e+00, -9.6833e-01,  1.5131e+00, -3.7041e+00, -7.7749e-02,
   1.5029e+00, -1.7764e+00,  1.7324e+00,  1.6241e+00,  2.6455e-01,
  -3.0840e+00,  7.5715e-01, -1.2903e+00,  2.3571e+00, -3.8793e+00,
   7.7635e-01,  3.9372e+00,  3.9900e-01, -6.8284e-01, -1.4018e+00,
  -2.1673e+00, -1.9244e+00,  1.0629e+00,  3.3378e-01, -8.3864e-01,
  -2.5646e-01, -1.7198e+00, -5.4607e-02, -1.4614e+00,  1.3352e+00,
  -1.8177e+00,  1.7254e+00,  4.9624e-01,  1.1314e+00, -1.5295e+00,
  -8.8629e-01, -2.7562e-01,  7.1799e-01,  1.5554e-01,  3.4230e+00,
   2.7167e+00,  1.1793e+00,  2.0961e-01,  3.3121e-01,  1.2322e+00,
   1.4375e+00, -4.2099e-01,  6.2814e-01, -1.9051e+00,  3.0593e-02,
   6.1895e-01, -3.1495e-01, -2.0444e-04,  2.2073e+00,  3.8856e-01,
   1.6554e+00,  1.1932e+00,  2.6678e+00, -5.5454e-01, -1.2078e+00,
   1.5709e-01, -1.1324e+00, -2.0163e+00,  1.4567e+00, -2.4244e-01,
  -1.9425e+00,  8.3090e-01,  1.7428e-01,  9.1676e-01,  8.8830e-03,
   2.4857e-01, -1.2018e+00, -2.3073e+00,  2.2553e+00, -1.5853e+00,
  -5.8452e-01,  9.2523e-01, -2.7129e-01, -7.6348e-01,  1.3506e+00,
   1.7429e+00,  3.0469e+00,  1.9319e+00, -2.6099e+00,  1.8484e+00,
   1.3795e+00,  2.0948e+00,  1.1545e+00, -2.9681e+00, -5.0455e-02,
  -5.3864e-01,  2.4820e+00, -1.1131e+00, -2.1827e-01, -2.7559e+00,
  -4.4502e-01, -2.8897e+00,  1.7430e+00, -1.5742e+00,  5.7160e-02,
   2.4764e+00, -2.5828e+00,  9.3866e-01, -1.3150e+00,  2.3863e+00,
   6.1536e-01,  1.7656e-01,  2.0245e+00,  1.6807e-01, -1.2850e+00,
   1.6425e-01,  1.7782e+00, -3.2221e+00,  6.1392e-01,  1.3269e+00,
  -3.1582e-02,  6.6331e-01, -6.8109e-01,  5.0985e-01, -4.2942e-01,
  -1.6438e-01,  7.9306e-01, -3.0776e+00,  1.8022e+00, -4.5356e-01,
  -1.6405e+00,  8.1761e-01,  1.4960e+00, -6.2266e-01,  8.5264e-01,
  -5.0226e-01, -1.5735e+00, -4.5090e+00, -5.0587e-01, -1.5471e+00,
  -5.3910e-01, -6.6574e-01,  7.6376e-01, -1.4926e+00, -7.8819e-01,
  -9.9256e-01,  1.1512e+00,  5.2091e-01,  1.6460e-01, -2.6747e+00,
  -1.7082e+00,  1.5789e+00, -2.8982e-01, -1.2842e+00, -1.1286e+00,
   7.6392e-01,  3.2199e+00,  7.5850e-01,  1.3628e+00, -1.3231e+00,
   2.2350e-02, -2.5602e+00,  6.7751e-01,  4.0511e-01,  1.8997e+00,
  -1.1051e+00, -1.3014e-01,  7.2024e-01,  6.2354e-02,  1.1913e-01,
  -1.1978e+00, -1.5625e+00, -2.5975e-01,  2.5911e+00, -3.2413e+00,
  -3.8988e-01, -4.0542e-01, -1.8894e+00,  3.4278e+00, -3.3625e-01,
  -2.0979e+00,  1.3275e+00, -2.0514e+00,  2.4583e-01, -7.3326e-01,
  -2.3684e+00,  2.8493e+00, -5.2075e-01,  2.2708e-01, -6.8701e-01,
  -7.0855e-01, -7.5334e-01,  7.3050e-02,  2.2246e+00, -2.6824e-01,
  -2.8289e-01, -1.8230e+00,  2.2047e+00, -2.4848e-01, -2.3042e-02,
   1.0358e+00, -2.7074e-01, -5.6816e-02, -9.1017e-01,  1.2943e-01,
  -1.4274e+00, -3.6128e-01,  7.3127e-01,  2.0264e+00,  7.2928e-01,
   1.7298e+00,  1.1075e+00, -7.0250e-01,  1.6928e+00,  2.0074e+00,
  -7.5464e-01,  1.6378e+00,  3.5970e-01, -2.2128e-01, -1.7607e-01,
   1.8260e+00, -2.5962e-01, -1.4320e+00,  7.8332e-01,  2.1438e+00,
  -2.4723e+00, -1.4913e-01,  6.2585e-01,  6.6819e-01,  2.3947e+00,
  -2.7173e+00,  2.4134e-03, -8.6530e-01, -9.7728e-01, -2.9815e+00,
   1.6895e+00, -7.1146e-01,  3.2025e+00, -9.4129e-01, -1.9695e+00,
   7.7711e-01, -3.2278e-01, -1.3727e+00,  2.9276e+00, -1.5440e-01,
   1.7169e+00,  5.5736e-01,  1.4620e-01, -1.1244e+00, -2.4633e+00,
  -2.2685e+00,  1.2459e+00, -2.0362e+00, -4.8331e-01, -6.3194e-01,
  -2.4082e+00, -9.0132e-01,  3.0541e+00, -2.2632e+00, -3.7800e-01,
  -3.1647e-01,  1.0785e+00, -3.0444e-01,  1.2112e+00, -1.3496e+00,
   1.0599e+00,  4.2607e-01,  4.0194e-01, -2.8586e+00,  1.0107e+00,
   1.5924e+00, -5.1770e-01,  1.3246e+00,  3.2268e-01, -1.3978e-01,
  -2.1841e+00,  1.6548e+00,  1.3903e+00,  6.3376e-01, -4.7083e-01,
   6.8377e-01, -1.3031e+00, -1.3292e-01, -1.1567e+00,  5.3419e-01,
```

```
-1.3412e+00, -1.5887e+00, -9.4468e-01, -2.4031e+00, 3.1785e+00,
1.1524e+00, -1.1699e+00, 9.8752e-01, -1.0660e+00, -2.1852e+00,
-3.1228e-01, 3.0012e+00, -1.2234e+00, 5.7454e-01, -2.1885e-01],
dtype=float32)
```

Representing documents using word embeddings

- Assuming that we have reasonable representations of words.
- How do we represent meaning of paragraphs or documents?
- Two simple approaches
 - Averaging embeddings
 - Concatenating embeddings

Averaging embeddings

All empty promises

$$(embedding(all) + embedding(empty) + embedding(promise))/3$$

Average embeddings with spaCy

- We can do this conveniently with [spaCy \(<https://spacy.io/usage/linguistic-features#vectors-similarity>\)](https://spacy.io/usage/linguistic-features#vectors-similarity).
- We need `en_core_web_md` model to access word vectors.
- You can download the model by going to command line and in your course `conda` environment and download `en_core_web_md` as follows.

```
conda activate cpsc330
python -m spacy download en_core_web_md
```

We can access word vectors for individual words in `spaCy` as follows.

In [24]: 1 `nlp("empty").vector[0:10]`

Out[24]: `array([0.010289, 4.9203 , -0.48081 , 3.5738 , -2.2516 , 2.1697 ,
-1.0116 , 2.4216 , -3.7343 , 3.3025], dtype=float32)`

We can get average embeddings for a sentence or a document in `spaCy` as follows:

In [25]:

```

1 s = "All empty promises"
2 doc = nlp(s)
3 avg_sent_emb = doc.vector
4 print(avg_sent_emb.shape)
5 print("Vector for: {} \n {}".format(s, (avg_sent_emb[0:10])))

```

(300,)

Vector for: All empty promises

-0.459937	1.9785299	1.0319	1.5123	1.4806334	2.73183
1.204	1.1724668	-3.5227966	-0.05656664		

Similarity between documents

- We can also get similarity between documents as follows.
- Note that this is based on average embeddings of each sentence.

In [80]:

```

1 doc1 = nlp("Deep learning is very popular these days.")
2 doc2 = nlp("Machine learning is dominated by neural networks.")
3 doc3 = nlp("A home-made fresh bread with butter and cheese.")
4
5 # Similarity of two documents
6 print(doc1, "<->", doc2, doc1.similarity(doc2))
7 print(doc2, "<->", doc3, doc2.similarity(doc3))

```

Deep learning is very popular these days. <-> Machine learning is dominated by neural networks. 0.6998688700400709

Machine learning is dominated by neural networks. <-> A home-made fresh bread with butter and cheese. 0.5098292895236649

- Do these scores make sense?
- There are no common words, but we are still able to identify that doc1 and doc2 are more similar than doc2 and doc3.
- You can use such average embedding representation in text classification tasks.

Airline sentiment analysis using average embedding representation

- Let's try average embedding representation for airline sentiment analysis.
- You can download the dataset [here](https://www.kaggle.com/jaskarancr/airline-sentiment-dataset) (<https://www.kaggle.com/jaskarancr/airline-sentiment-dataset>).

In [27]:

```
1 df = pd.read_csv("../data/Airline-Sentiment-2-w-AA.csv", encoding="ISO-8859-1")
```

```
In [28]: 1 from sklearn.model_selection import cross_validate, train_test_split
2
3 train_df, test_df = train_test_split(df, test_size=0.2, random_state=12)
4 X_train, y_train = train_df["text"], train_df["airline_sentiment"]
5 X_test, y_test = test_df["text"], test_df["airline_sentiment"]
```

```
In [29]: 1 train_df.head()
```

Out[29]:

	_unit_id	_golden	_unit_state	_trusted_judgments	_last_judgment_at	airline_sentiment	air
5789	681455792	False	finalized	3	2/25/15 4:21	negative	
8918	681459957	False	finalized	3	2/25/15 9:45	neutral	
11688	681462990	False	finalized	3	2/25/15 9:53	negative	
413	681448905	False	finalized	3	2/25/15 10:10	neutral	
4135	681454122	False	finalized	3	2/25/15 10:08	negative	

Bag-of-words representation for sentiment analysis

```
In [30]: 1 pipe = make_pipeline(
2     CountVectorizer(stop_words="english"), LogisticRegression(max_iter=
3 )
4 pipe.named_steps["countvectorizer"].fit(X_train)
5 X_train_transformed = pipe.named_steps["countvectorizer"].transform(X_t
6 print("Data matrix shape:", X_train_transformed.shape)
7 pipe.fit(X_train, y_train);
```

Data matrix shape: (11712, 13064)

```
In [31]: 1 print("Train accuracy {:.2f}".format(pipe.score(X_train, y_train)))
2 print("Test accuracy {:.2f}".format(pipe.score(X_test, y_test)))
```

Train accuracy 0.94
Test accuracy 0.80

Sentiment analysis with average embedding representation

- Let's see how we can get word vectors using spaCy.
- Let's create average embedding representation for each example.

```
In [32]: 1 X_train_embeddings = pd.DataFrame([text.vector for text in nlp.pipe(X_t
2 X_test_embeddings = pd.DataFrame([text.vector for text in nlp.pipe(X_te
```

We have reduced dimensionality from 13,064 to 300!

```
In [33]: 1 X_train_embeddings.shape
```

Out[33]: (11712, 300)

```
In [34]: 1 X_train_embeddings.head()
```

	0	1	2	3	4	5	6	7	8	
0	1.259942	4.856640	-2.677500	-1.875390	0.459240	-0.304300	3.273020	2.008458	-4.818360	1
1	-0.563826	0.869816	-2.462877	0.057751	0.892089	0.566698	-0.283121	3.594112	-1.578107	1
2	-0.707503	0.908782	-3.248327	-0.667797	2.590958	2.246804	-1.095754	3.953429	-1.524224	0
3	-0.977736	4.676425	-0.224362	-1.011286	3.981441	-1.132660	0.988456	2.997113	0.553975	-0
4	-0.725984	0.178514	-1.163662	0.597000	4.621603	-1.166608	1.256955	3.940082	0.530063	-0

5 rows × 300 columns

Sentiment classification using average embeddings

- What are the train and test accuracies with average word embedding representation?
- The accuracy is a bit better with less overfitting.
- Note that we are using **transfer learning** here.
- The embeddings are trained on a completely different corpus.

```
In [35]: 1 lgr = LogisticRegression(max_iter=1000, C=.1)
2 lgr.fit(X_train_embeddings, y_train)
3 print("Train accuracy {:.2f}".format(lgr.score(X_train_embeddings, y_tr
4 print("Test accuracy {:.2f}".format(lgr.score(X_test_embeddings, y_te
```

Train accuracy 0.80
Test accuracy 0.79

Sentiment classification using advanced sentence representations

- Since representing documents is so essential for text classification tasks, there are more advanced methods for document representation.

We will use: `conda install -c conda-forge sentence-transformers`

```
In [36]: 1 from sentence_transformers import SentenceTransformer
          2
          3 embedder = SentenceTransformer("sentence-transformers/paraphrase-distil
```

```
In [37]: 1 emb_sents = embedder.encode("all empty promises")
          2 emb_sents.shape
```

Out[37]: (768,)

```
In [38]: 1 emb_train = embedder.encode(train_df["text"].tolist())
          2 emb_train_df = pd.DataFrame(emb_train, index=train_df.index)
          3 emb_train_df
```

	0	1	2	3	4	5	6	7	
5789	-0.120494	0.250263	-0.022795	-0.116368	0.078650	0.037357	-0.251341	0.321429	-0.1439
8918	-0.182954	0.118282	0.066341	-0.136098	0.094947	-0.121303	0.069233	-0.097500	0.0251
11688	-0.032988	0.630251	-0.079516	0.148981	0.194708	-0.226263	-0.043630	0.217398	-0.0107
413	-0.119258	0.172168	0.098697	0.319858	0.415475	0.248359	-0.025923	0.385350	0.0664
4135	0.094240	0.360193	0.213747	0.363690	0.275521	0.134936	-0.276319	0.009336	-0.0215
...
5218	-0.204408	-0.145289	-0.064201	0.213571	-0.140225	0.338556	-0.148578	0.224515	-0.0429
12252	0.108408	0.438293	0.216812	-0.349289	0.422689	0.377760	0.045198	-0.034095	0.4278
1346	0.068411	0.017591	0.236154	0.221446	-0.103568	0.055510	0.062910	0.067424	-0.0038
11646	-0.091488	-0.155708	0.032391	0.018314	0.524997	0.563933	-0.080985	0.097982	-0.5352
3582	0.185626	0.092904	0.097085	-0.174650	-0.193584	0.047294	0.098216	0.332670	0.1630

11712 rows × 768 columns

In [39]:

```

1 emb_test = embedder.encode(test_df["text"].tolist())
2 emb_test_df = pd.DataFrame(emb_test, index=test_df.index)
3 emb_test_df

```

Out[39]:

	0	1	2	3	4	5	6	7
1671	-0.002864	0.217326	0.124349	-0.082548	0.709688	-0.582441	0.257897	0.169356
10951	-0.141048	0.137934	0.131319	0.194773	0.868204	0.078791	-0.131656	0.036244
5382	-0.252943	0.527507	-0.065608	0.013467	0.207989	0.003881	-0.066281	0.253166
3954	0.054319	0.096738	0.113037	0.032039	0.493064	-0.641102	0.078760	0.402187
11193	-0.065858	0.223270	0.507333	0.266193	0.104696	-0.219555	0.146247	0.315649
...
5861	0.077512	0.322276	0.026697	-0.111393	0.174207	0.235201	0.053888	0.244942
3627	-0.173311	-0.023604	0.190388	-0.136543	-0.360269	-0.444686	0.056311	0.291941
12559	-0.124635	-0.101799	0.129061	0.636907	0.681090	0.399300	-0.078321	0.221824
8123	0.063508	0.332506	0.119605	-0.001363	-0.161802	-0.082302	-0.025883	0.048027
210	0.015537	0.425568	0.350672	0.113120	-0.128615	0.098112	0.222081	0.101654

2928 rows × 768 columns

In [40]:

```

1 lgr = LogisticRegression(max_iter=1000)
2 lgr.fit(emb_train, y_train)
3 print("Train accuracy {:.2f}".format(lgr.score(emb_train, y_train)))
4 print("Test accuracy {:.2f}".format(lgr.score(emb_test, y_test)))

```

Train accuracy 0.87

Test accuracy 0.83

- Some improvement over bag of words and average embedding representations!
- But much slower ...

Break (5 min)

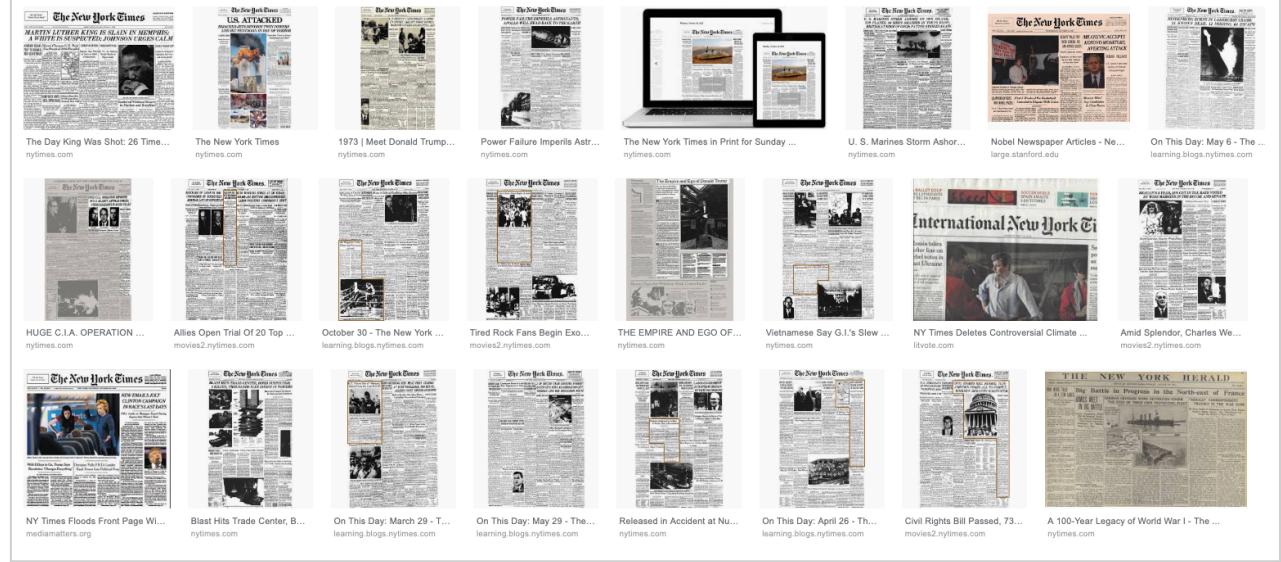


Topic modeling

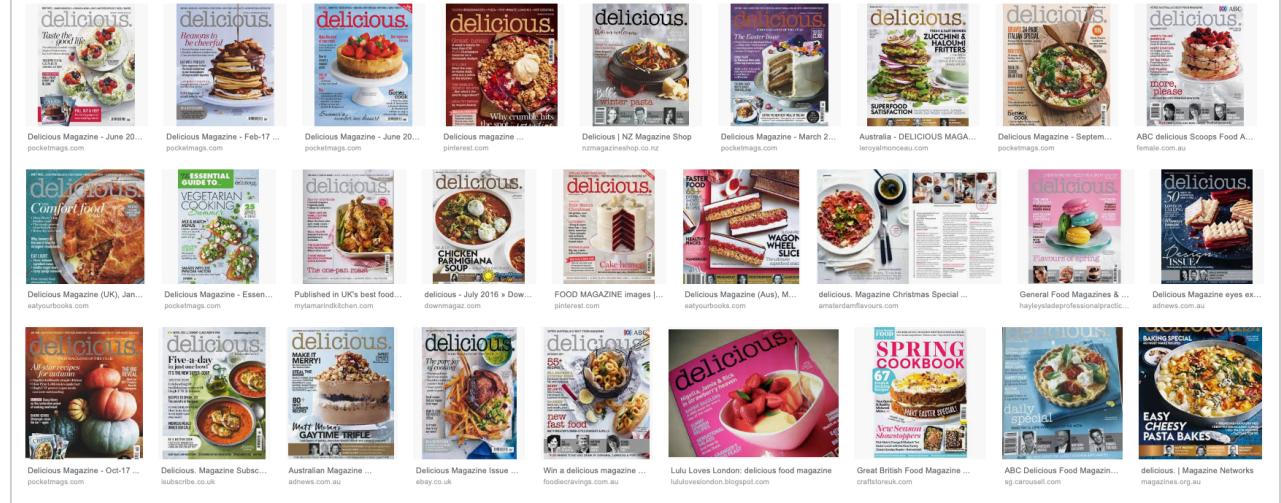
Topic modeling motivation

- Suppose you have a large collection of documents on a variety of topics.

Example: A corpus of news articles



Example: A corpus of food magazines



A corpus of scientific articles

Poisoning by ice-cream.

No chemist certainly would suppose that the same poison exists in all samples of ice-cream which have produced untoward symptoms in man. Mineral poisons, copper, lead, arsenic, and mercury, have all been found in ice cream. In some instances these have been used with criminal intent. In other cases their presence has been accidental. Likewise, that vanilla is sometimes the bearer, at least, of the poison, is well known to all chemists. Dr. Bartley's idea that the poisonous properties of the cream which he examined were due to putrid gelatine is certainly a rational theory. The poisonous principle might in this case arise from the decomposition of the gelatine; or with the gelatine there may be introduced into the milk a ferment, by the growth of which a poison is produced.

But in the cream which I examined, none of the above sources of the poisoning existed. There were no mineral poisons present. No gelatine of any kind had been used in making the cream. The vanilla used was shown to be not poisonous. This showing was made, not by a chemical analysis, which might not have been conclusive, but Mr. Novie and I drank of the vanilla extract which was used, and no ill results followed. Still, from this cream we isolated the same poison which I had before found in poisonous cheese (*Zeitschrift für physiologische Chemie*, X,

RNA Editing and the Evolution of Parasites

Larry Simpson and Dmitri A. Maslov

The kinetoplastid flagellates, together with the trypanosomes, represent the earliest extant lineage of eukaryotic organisms containing mitochondria (1). Within the kinetoplastids, there are two major groups, the poorly resolved two-circular DNA (2cDNA) group and the group of free-living and parasitic cells and the better known trypanosomatids, which are obligate intracellular parasites (2).

Perhaps because of the antiquity of the trypanosomal lineage, these cells possess several unique genetic features (see review by Hedges and Barker, perspective by Nilzen)—one of which is RNA editing of mitochondrial genes. The mitochondrial RNA editing function (3–7) creates open reading frames in regions of insertion (or occasional deletion) of uridine (U) residues at a few specific sites within the coding region of the tRNA^{Asp} (5'-editing) or at multiple specific sites throughout the mRNA (non-editing). The

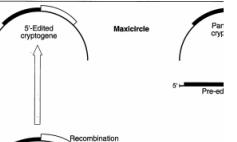
trial, there is disagreement on the nature of the host. The "vavender first" model (1) holds that the initial parasitism was in the gut of pre-Cambrian invertebrates. Coevolution of parasitic and host would lead to a wide range of interactions that contribute any natural selection pressure on the parasite pathways. In this theory, diagenetic life cycles (alternating invertebrate and vertebrate hosts) provide the mechanism for the acquisition by some hemiparasites and diplovers of the ability to feed on the blood

Ecologists have known since the pioneering work of May in the mid-1970's (1) that the population dynamics of animals and plants can be extremely complex. This complexity arises from two sources: The tangled web of interactions that constitute any natural community provides many complex pathways for species to interact, both directly and indirectly. And even in isolated populations the nonlinear feedback processes present in all natural populations can result in dramatic behavioral oscillations. Such populations can show sustained oscillatory dynamics and chaos, the latter characterized by unpredictable, seemingly random fluctuations of natural resources. On page 389 of this issue, Costantino et al. (2) provide the

Chaotic Beetles
Charles Godfray and Michael Hassell

move over the surface of the attractor, sets of adjacent trajectories pull apart, then spread and merge, so it is impossible to predict exact population densities into the future. The strength of the mixing that occurs in the population is related to initial conditions can be measured mathematically estimating the Lyapunov exponent, which measures the rate of divergence of the trajectories. Chaotic dynamics are often deterministic and nonperiodic. They have been mathematically estimated as attractor dimension and Lapunov exponent. In time series analysis, some chaotic populations have been identified (some insects, rodents, and birds). Interestingly, human childhood diseases, but also some antibiotic resistance, exhibit chaotic dynamics, which are broader generalization (3).

An alternative approach is to determine population models from natural populations and then compare them with their predictions with the dynamics in the field. This technique has been applied to insect populations, helping statistical advances in parameter estimation. Good



The authors are in the Department of Biology, Imperial College at Silwood Park, Ascot, Berks, SL5 2PU, UK. E-mail: m.hassell@imperial.ac.uk

SCIENCE • VOL. 275 • 17 JANUARY 1997

(Credit: [Dave Blei's presentation \(\[http://www.cs.columbia.edu/~blei/talks/Blei_Science_2008.pdf\]\(http://www.cs.columbia.edu/~blei/talks/Blei_Science_2008.pdf\)\)](http://www.cs.columbia.edu/~blei/talks/Blei_Science_2008.pdf)

Topic modeling motivation

- Humans are pretty good at reading and understanding a document and answering questions such as
 - What is it about?
 - Which documents is it related to?
- But for a large collection of documents it would take years to read all documents and organize and categorize them so that they are easy to search.
- You need an automated way
 - to get an idea of what's going on in the data or
 - to pull documents related to a certain topic

Topic modeling

- Topic modeling gives you an ability to summarize the major themes in a large collection of documents (corpus).
 - Example: The major themes in a collection of news articles could be
 - politics
 - entertainment
 - sports
 - technology
 - ...
- A common tool to solve such problems is unsupervised ML methods.
- Given the hyperparameter K , the idea of topic modeling is to describe the data using K "topics"

Topic modeling: Input and output

- Input
 - A large collection of documents
 - A value for the hyperparameter K (e.g., $K = 3$)
- Output
 1. Topic-words association
 - For each topic, what words describe that topic?
 2. Document-topics association
 - For each document, what topics are expressed by the document?

Topic modeling: Example

- Topic-words association
 - For each topic, what words describe that topic?
 - A topic is a mixture of words.

TOPIC 1

probabilistic,
bayesian, markov,
stationary, model,
hidden, word2vec,

TOPIC 2

fashion, clothes,
model, elegant,
style, pattern,
fresh, designer,

TOPIC 3

nutrition, health,
butter, soup,
bread, baking,
apple, fresh,

Topic modeling: Example

- Document-topics association
 - For each document, what topics are expressed by the document?
 - A document is a mixture of topics.

	TOPIC 1	TOPIC 2	TOPIC 3
Document 1	10%	30%	60%
Document 2	100%	0%	0%
Document 3	0%	100%	0%
Document 4	30%	30%	40%
...

Topic modeling: Input and output

- Input
 - A large collection of documents
 - A value for the hyperparameter K (e.g., $K = 3$)
- Output
 - For each topic, what words describe that topic?
 - For each document, what topics are expressed by the document?

TOPIC 1

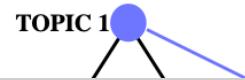
probabilistic,
bayesian, markov,
stationary, model,
hidden, word2vec,
pattern, ...

TOPIC 2

fashion, clothes,
model, elegant,
style, pattern,
fresh, designer,
creative, ...

TOPIC 3

nutrition, health,
butter, soup,
bread, baking,
apple, fresh,
creative, ...



TOPIC 1 TOPIC 2 TOPIC 3

Topic modeling: Some applications

- Topic modeling is a great EDA tool to get a sense of what's going on in a large corpus.
- Some examples
 - If you want to pull documents related to a particular lawsuit.
 - You want to examine people's sentiment towards a particular candidate and/or political party and so you want to pull tweets or Facebook posts related to election.

Topic modeling toy example

```
In [41]: 1 toy_df = pd.read_csv("../data/toy_lda_data.csv")
          2 toy_df
```

Out[41]:

	doc_id	text
0	1	famous fashion model
1	2	fashion model pattern
2	3	fashion model probabilistic topic model confer...
3	4	famous fashion model
4	5	fresh fashion model
5	6	famous fashion model
6	7	famous fashion model
7	8	famous fashion model
8	9	famous fashion model
9	10	creative fashion model
10	11	famous fashion model
11	12	famous fashion model
12	13	fashion model probabilistic topic model confer...
13	14	probabilistic topic model
14	15	probabilistic model pattern
15	16	probabilistic topic model
16	17	probabilistic topic model
17	18	probabilistic topic model
18	19	probabilistic topic model
19	20	probabilistic topic model
20	21	probabilistic topic model
21	22	fashion model probabilistic topic model confer...
22	23	apple kiwi nutrition
23	24	kiwi health nutrition
24	25	fresh apple health
25	26	probabilistic topic model
26	27	creative health nutrition
27	28	probabilistic topic model
28	29	probabilistic topic model
29	30	hidden markov model probabilistic
30	31	probabilistic topic model
31	32	probabilistic topic model
32	33	apple kiwi nutrition
33	34	apple kiwi health
34	35	apple kiwi nutrition

doc_id		text
35	36	fresh kiwi health
36	37	apple kiwi nutrition
37	38	apple kiwi nutrition
38	39	apple kiwi nutrition

```
In [42]: 1 from gensim import corpora, matutils, models
2
3 corpus = [doc.split() for doc in toy_df["text"].tolist()]
# Create a vocabulary for the lda (Latent Dirichlet Allocation) model
4 dictionary = corpora.Dictionary(corpus)
# Convert our corpus into document-term matrix for Lda
5 doc_term_matrix = [dictionary.doc2bow(doc) for doc in corpus]
```

```
In [43]: 1 from gensim.models import LdaModel
2
3 # Train an lda model
4 lda = models.LdaModel(
5     corpus=doc_term_matrix,
6     id2word=dictionary,
7     num_topics=3,
8     random_state=123,
9     passes=10,
10 )
```

```
In [44]: 1 ### Examine the topics in our LDA model
2 lda.print_topics(num_words=4)
```

```
Out[44]: [(0,
    '0.303*"model" + 0.296*"probabilistic" + 0.261*"topic" + 0.040*"pattern"),
(1, '0.245*"kiwi" + 0.219*"apple" + 0.218*"nutrition" + 0.140*"health"),
(2, '0.308*"fashion" + 0.307*"model" + 0.180*"famous" + 0.071*"conference")]
```

```
In [45]: 1 ### Examine the topic distribution for a document
2 print("Document: ", corpus[0])
3 df = pd.DataFrame(lda[doc_term_matrix[0]], columns=["topic id", "probability"])
4 df.sort_values("probability", ascending=False)
```

```
Document: ['famous', 'fashion', 'model']
```

```
Out[45]: topic id  probability
```

2	2	0.828760
0	0	0.087849
1	1	0.083391

You can also visualize the topics using pyLDAvis .

```
pip install pyLDAvis
```

Do not install it using `conda`. They have made some changes in the recent version and `conda` build is not available for this version yet.

In [46]:

```
1 # Visualize the topics
2 import pyLDAvis
3
4 pyLDAvis.enable_notebook()
5 import pyLDAvis.gensim_models as gensimvis
6
7 vis = gensimvis.prepare(lda, doc_term_matrix, dictionary, sort_topics=False)
8 vis
```

huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:

- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)

huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:

- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)

huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:

- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)

huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:

- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)

huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:

- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)

huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:

- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)

huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:

- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)

huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:

- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)

huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:

- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)

- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
 - Avoid using `tokenizers` before the fork if possible
 - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
 - Avoid using `tokenizers` before the fork if possible
 - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
 - Avoid using `tokenizers` before the fork if possible
 - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)

Out[46]:

Topic modeling pipeline

- Preprocess your corpus.
- Train LDA using Gensim.
- Interpret your topics.

Data

In [47]:

```

1 import wikipedia
2
3 queries = [
4     "Artificial Intelligence",
5     "unsupervised learning",
6     "Supreme Court of Canada",
7     "Peace, Order, and Good Government",
8     "Canadian constitutional law",
9     "ice hockey",
10    ]
11 wiki_dict = {"wiki query": [], "text": []}
12 for i in range(len(queries)):
13     wiki_dict["text"].append(wikipedia.page(queries[i]).content)
14     wiki_dict["wiki query"].append(queries[i])
15
16 wiki_df = pd.DataFrame(wiki_dict)
17 wiki_df

```

Out[47]:

	wiki query	text
0	Artificial Intelligence	Artificial intelligence (AI) is intelligence—p...
1	unsupervised learning	Supervised learning (SL) is a machine learning...
2	Supreme Court of Canada	The Supreme Court of Canada (SCC; French: Cour...
3	Peace, Order, and Good Government	In many Commonwealth jurisdictions, the phrase...
4	Canadian constitutional law	Canadian constitutional law (French: droit con...
5	ice hockey	Ice hockey (or simply hockey) is a team sport ...

Preprocessing the corpus

- **Preprocessing is crucial!**
- Tokenization, converting text to lower case
- Removing punctuation and stopwords
- Discarding words with length < threshold or word frequency < threshold
- Possibly lemmatization: Consider the lemmas instead of inflected forms.
- Depending upon your application, restrict to specific part of speech;
 - For example, only consider nouns, verbs, and adjectives

We'll use `spaCy` (<https://spacy.io/>) for preprocessing.

In [48]:

```

1 import spacy
2
3 nlp = spacy.load("en_core_web_md", disable=["parser", "ner"])

```

In [49]:

```

1 def preprocess(
2     doc,
3     min_token_len=2,
4     irrelevant_pos=["ADV", "PRON", "CCONJ", "PUNCT", "PART", "DET", "AD"
5     ):
6     """
7         Given text, min_token_len, and irrelevant_pos carry out preprocessing
8         and return a preprocessed string.
9
10    Parameters
11    -----
12    doc : (spaCy doc object)
13        the spaCy doc object of the text
14    min_token_len : (int)
15        min_token_length required
16    irrelevant_pos : (list)
17        a list of irrelevant pos tags
18
19    Returns
20    -----
21    (str) the preprocessed text
22    """
23
24    clean_text = []
25
26    for token in doc:
27        if (
28            token.is_stop == False # Check if it's not a stopword
29            and len(token) > min_token_len # Check if the word meets min length
30            and token.pos_ not in irrelevant_pos
31        ): # Check if the POS is in the acceptable POS tags
32            lemma = token.lemma_ # Take the lemma of the word
33            clean_text.append(lemma.lower())
34    return " ".join(clean_text)

```

In [50]:

```
1 wiki_df["text_pp"] = [preprocess(text) for text in nlp.pipe(wiki_df["te
```

In [51]:

```
1 wiki_df
```

Out[51]:

	wiki query	text	text_pp
0	Artificial Intelligence	Artificial intelligence (AI) is intelligence —p...	artificial intelligence intelligence perceive ...
1	unsupervised learning	Supervised learning (SL) is a machine learning...	supervised learning machine learn paradigm pro...
2	Supreme Court of Canada	The Supreme Court of Canada (SCC; French: Cour...	supreme court canada scc french cour suprême c...
3	Peace, Order, and Good Government	In many Commonwealth jurisdictions, the phrase...	commonwealth jurisdiction phrase peace order g...
4	Canadian constitutional law	Canadian constitutional law (French: droit con...	canadian constitutional law french droit const...
5	ice hockey	Ice hockey (or simply hockey) is a team sport ...	ice hockey hockey team sport play ice skate ic...

Training LDA with gensim (<https://radimrehurek.com/gensim/models/Idamodel.html>)

To train an LDA model with [gensim](https://radimrehurek.com/gensim/models/Idamodel.html) (<https://radimrehurek.com/gensim/models/Idamodel.html>), you need

- Document-term matrix
- Dictionary (vocabulary)
- The number of topics (K): num_topics
- The number of passes: passes

Gensim's doc2bow

- Let's first create a dictionary using [corpora.Dictionary](https://radimrehurek.com/gensim/corpora/dictionary.html) (<https://radimrehurek.com/gensim/corpora/dictionary.html>).

```
In [52]: 1 corpus = [doc.split() for doc in wiki_df["text_pp"].tolist()]
2 dictionary = corpora.Dictionary(corpus) # Create a vocabulary for the
3 pd.DataFrame(
4     dictionary.token2id.keys(), index=dictionary.token2id.values(), col
5 )
```

Out[52]:

	Word
0	1.6
1	134,777
2	167,038
3	1863
4	1943
...	...
3880	wrist
3881	yale
3882	york
3883	youth
3884	zhenskaya

3885 rows × 1 columns

Gensim's doc2bow

- Now let's convert our corpus into document-term matrix for LDA using [dictionary.doc2bow](https://radimrehurek.com/gensim/corpora/dictionary.html#gensim.corpora.dictionary.Dictionary) (<https://radimrehurek.com/gensim/corpora/dictionary.html#gensim.corpora.dictionary.Dictionary>).

- For each document, it stores the frequency of each token in the document in the format `(token_id, frequency)`

```
In [53]: 1 doc_term_matrix = [dictionary.doc2bow(doc) for doc in corpus]
2 doc_term_matrix[1][:20]
```

```
Out[53]: [(57, 4),
(68, 3),
(74, 1),
(80, 3),
(88, 1),
(89, 3),
(114, 53),
(123, 1),
(132, 4),
(133, 1),
(134, 1),
(146, 1),
(147, 1),
(148, 4),
(149, 3),
(164, 1),
(166, 1),
(174, 2),
(182, 2),
(192, 4)]
```

Now we are ready to train an LDA model.

```
In [54]: 1 from gensim.models import LdaModel
2
3 num_topics = 3
4
5 lda = models.LdaModel(
6     corpus=doc_term_matrix,
7     id2word=dictionary,
8     num_topics=num_topics,
9     random_state=42,
10    passes=10,
11 )
```

Examine the topics and topic distribution for a document in our LDA model

```
In [55]: 1 lda.print_topics(num_words=4) # Topics
```

```
Out[55]: [(0, '0.002*"hockey" + 0.001*"ice" + 0.001*"game" + 0.001*"league"'),
(1,
 '0.008*"algorithm" + 0.008*"intelligence" + 0.007*"learning" + 0.007
 *"\\"displaystyle"'),
(2, '0.029*"hockey" + 0.018*"ice" + 0.014*"team" + 0.014*"player"')]
```

```
In [56]: 1 print("Document: ", wiki_df.iloc[0][0])
2 print("Topic assignment for document: ", lda[doc_term_matrix[0]]) # To
```

Document: Artificial Intelligence
Topic assignment for document: [(1, 0.9998414)]

Visualize topics

```
In [57]: 1 vis = gensimvis.prepare(lda, doc_term_matrix, dictionary, sort_topics=False)
2 vis
```

Out[57]:

(Optional) Topic modeling with sklearn

- We are using Gensim LDA so that we'll be able to use CoherenceModel to evaluate topic model later.
- But we can also train an LDA model with sklearn .

```
In [58]: 1 from sklearn.feature_extraction.text import CountVectorizer
2
3 vec = CountVectorizer()
4 X = vec.fit_transform(wiki_df["text_pp"])
```

```
In [59]: 1 from sklearn.decomposition import LatentDirichletAllocation
2
3 n_topics = 3
4 lda = LatentDirichletAllocation(
5     n_components=n_topics, learning_method="batch", max_iter=10, random_
6 )
7 document_topics = lda.fit_transform(X)
```

```
In [60]: 1 print("lda.components_.shape: {}".format(lda.components_.shape))
```

lda.components_.shape: (3, 3814)

```
In [61]: 1 sorting = np.argsort(lda.components_, axis=1)[:, ::-1]
2 feature_names = np.array(vec.get_feature_names())
```

In [65]:

```

1  ## No need to look at this code
2  def print_topics(topics, feature_names, sorting, topics_per_chunk=6,
3                   n_words=20):
4      for i in range(0, len(topics), topics_per_chunk):
5          # for each chunk:
6          these_topics = topics[i: i + topics_per_chunk]
7          # maybe we have less than topics_per_chunk left
8          len_this_chunk = len(these_topics)
9          # print topic headers
10         print(("topic {:<8}" * len_this_chunk).format(*these_topics))
11         print(("----- {0:<5}" * len_this_chunk).format(""))
12         # print top n_words frequent words
13         for i in range(n_words):
14             try:
15                 print(("{:<14}" * len_this_chunk).format(
16                     *feature_names[sorting[these_topics, i]]))
17             except:
18                 pass
19         print("\n")
20
21     print_topics(
22         topics=range(3),
23         feature_names=feature_names,
24         sorting=sorting,
25         topics_per_chunk=5,
26         n_words=10,
27     )
28

```

topic 0	topic 1	topic 2
-----	-----	-----
hockey	court	displaystyle
ice	law	algorithm
team	intelligence	function
player	provincial	training
game	government	learn
play	power	learning
league	federal	datum
penalty	canada	feature
puck	artificial	input
canada	justice	supervised

Basic text preprocessing

Introduction

- Why do we need preprocessing?
 - Text data is unstructured and messy.
 - We need to "normalize" it before we do anything interesting with it.
- Example:
 - **Lemma:** Same stem, same part-of-speech, roughly the same meaning
 - Vancouver's → Vancouver
 - computers → computer
 - rising → rise, rose, rises

Tokenization

- Sentence segmentation
 - Split text into sentences
- Word tokenization
 - Split sentences into words

Tokenization: sentence segmentation

MDS is a Master's program at UBC in British Columbia. MDS teaching team is truly multicultural!! Dr. George did his Ph.D. in Scotland. Dr. Timbers, Dr. Ostblom, Dr. Rodríguez-Arelis, and Dr. Kolhatkar did theirs in Canada. Dr. Gelbart did his PhD in the U.S.

- How many sentences are there in this text?

```
In [66]: 1 ##### Let's do sentence segmentation on "."
2 text = (
3     "MDS is a Master's program at UBC in British Columbia. "
4     "MDS teaching team is truly multicultural!! "
5     "Dr. George did his Ph.D. in Scotland. "
6     "Dr. Timbers, Dr. Ostblom, Dr. Rodríguez-Arelis, and Dr. Kolhatkar
7     "Dr. Gelbart did his PhD in the U.S."
8 )
9
10 print(text.split("."))
```

```
[ "MDS is a Master's program at UBC in British Columbia", ' MDS teaching t
eam is truly multicultural!! Dr', ' George did his Ph', 'D', ' in Scotlan
d', ' Dr', ' Timbers, Dr', ' Ostblom, Dr', ' Rodríguez-Arelis, and Dr', ' '
Kolhatkar did theirs in Canada', ' Dr', ' Gelbart did his PhD in the U',
'S', '' ]
```

Sentence segmentation

- In English, period (.) is quite ambiguous. (In Chinese, it is unambiguous.)
 - Abbreviations like Dr., U.S., Inc.
 - Numbers like 60.44%, 0.98
- ! and ? are relatively ambiguous.
- How about writing regular expressions?
- A common way is using off-the-shelf models for sentence segmentation.

In [67]:

```

1 ### Let's try to do sentence segmentation using nltk
2 from nltk.tokenize import sent_tokenize
3
4 sent_tokenized = sent_tokenize(text)
5 print(sent_tokenized)

```

["MDS is a Master's program at UBC in British Columbia.", 'MDS teaching t
eam is truly multicultural!!', 'Dr. George did his Ph.D. in Scotland.',
'Dr. Timbers, Dr. Ostblom, Dr. Rodríguez-Arelis, and Dr. Kolhatkar did th
eirs in Canada.', 'Dr. Gelbart did his PhD in the U.S.']

Word tokenization

MDS is a Master's program at UBC in British Columbia.

- How many words are there in this sentence?
- Is whitespace a sufficient condition for a word boundary?

Word tokenization

MDS is a Master's program at UBC in British Columbia.

- What's our definition of a word?
 - Should British Columbia be one word or two words?
 - Should punctuation be considered a separate word?
 - What about the punctuations in U.S. ?
 - What do we do with words like Master's ?
- This process of identifying word boundaries is referred to as **tokenization**.
- You can use regex but better to do it with off-the-shelf ML models.

In [68]:

```

1 ### Let's do word segmentation on white spaces
2 print("Splitting on whitespace: ", [sent.split() for sent in sent_tokenized])
3
4 ### Let's try to do word segmentation using nltk
5 from nltk.tokenize import word_tokenize
6
7 word_tokenized = [word_tokenize(sent) for sent in sent_tokenized]
8 # This is similar to the input format of word2vec algorithm
9 print("\n\n\nTokenized: ", word_tokenized)

```

Splitting on whitespace: [['MDS', 'is', 'a', "Master's", 'program', 'at', 'UBC', 'in', 'British', 'Columbia.'], ['MDS', 'teaching', 'team', 'is', 'truly', 'multicultural!!!'], ['Dr.', 'George', 'did', 'his', 'Ph.D.', 'in', 'Scotland.'], ['Dr.', 'Timbers', 'Dr.', 'Ostblom', 'Dr.', 'Rodríguez-Arelis', 'and', 'Dr.', 'Kolhatkar', 'did', 'theirs', 'in', 'Canada.'], ['Dr.', 'Gelbart', 'did', 'his', 'PhD', 'in', 'the', 'U.S.']]

Tokenized: [['MDS', 'is', 'a', 'Master', "'s", 'program', 'at', 'UBC', 'in', 'British', 'Columbia', '.'], ['MDS', 'teaching', 'team', 'is', 'truly', 'multicultural', '!', '!'], ['Dr.', 'George', 'did', 'his', 'Ph.D.', 'in', 'Scotland', '.'], ['Dr.', 'Timbers', 'Dr.', 'Ostblom', 'Dr.', 'Rodríguez-Arelis', 'and', 'Dr.', 'Kolhatkar', 'did', 'theirs', 'in', 'Canada', '.'], ['Dr.', 'Gelbart', 'did', 'his', 'PhD', 'in', 'the', 'U.S', '.']]

Word segmentation

For some languages you need much more sophisticated tokenizers.

- For languages such as Chinese, there are no spaces between words.
 - [jieba \(<https://github.com/fxsjy/jieba>\)](https://github.com/fxsjy/jieba) is a popular tokenizer for Chinese.
- German doesn't separate compound words.
 - Example: *rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz*
 - (the law for the delegation of monitoring beef labeling)

Types and tokens

- Usually in NLP, we talk about
 - **Type** an element in the vocabulary
 - **Token** an instance of that type in running text

Exercise for you

UBC is located in the beautiful province of British Columbia. It's very close to the U.S. border. You'll get to the USA border in about 45 mins by car.

- Consider the example above.

- How many types? (task dependent)
- How many tokens?

Other commonly used preprocessing steps

- Punctuation and stopword removal
- Stemming and lemmatization

Punctuation and stopword removal

- The most frequently occurring words in English are not very useful in many NLP tasks.
 - Example: *the*, *is*, *a*, and punctuation
- Probably not very informative in many tasks

In [69]:

```

1 # Let's use `nltk.stopwords`.
2 # Add punctuations to the list.
3 stop_words = list(set(stopwords.words("english")))
4 import string
5
6 punctuation = string.punctuation
7 stop_words += list(punctuation)
8 # stop_words.extend(['``', '``', 'br', "''", "", "'''", "'s"])
9 print(stop_words)

```

```

['s', 'won', 'to', "you're", "don't", 'any', 'now', "isn't", 'up', 'this',
 'too', 'under', 'for', 'd', 'themselves', 'in', 'hasn', "shan't", 'before',
 'doesn', 'your', 'here', 'out', 'such', 'during', 'and', 'were',
 'n't', "mightn't", 'then', 'about', 'couldn', 'if', 'being', 'don', 'does',
 'mightn', 'shouldn', 'same', 'wasn', "you've", 'own', 'isn', 'i', 'his',
 'just', 'all', 'the', 'at', 'when', 'can', "it's", 'against', 'on',
 'was', 'between', 'or', 'once', 'there', 'other', "aren't", 'we', 'hadn',
 'from', 'himself', 'needn', 'didn', 'these', 'her', 'nor', 'shan', 'me',
 'their', 'yourself', 'above', 'down', 'll', "wouldn't", "hasn't", "you'll",
 't', 'have', 'whom', 'those', "you'd", 'that', 'only', "won't", 'because',
 'a', 'why', 'each', 'as', 'aren', 'theirs', "wasn't", 'them', 'over',
 'very', 'myself', 'were', 'been', 'o', 'doing', 'more', 'ma', 'again',
 'ain', 'hers', 'so', 'through', 'both', "haven't", 'than', 'y', 'has',
 "didn't", 'm', 'itself', 'am', 'she', 'off', 'not', 'having', 'had',
 "doesn't", "that'll", 've', 'him', 'an', 'yourselves', 'while', 'after',
 'few', 'mustn', 'do', 'its', "couldn't", 'they', 'my', 'yours', 'but',
 'into', 'did', 'most', 'where', 'will', 'is', 'with', 'ourselves', 'who',
 'no', "she's", 'until', 'our', "should've", 'it', 'you', "needn't", 'how',
 'are', 'below', 'further', "hadn't", "wouldn", 'should', "shouldn't",
 'haven', 'what', 'which', 'by', 'some', 'weren', "mustn't", 'be', 're',
 'herself', 'ours', 'he', 'of', '!', '"', '#', '$', '%', '&', "'",
 '(', ')', '*', '+', ',', '-', '.', '/', ':', ';', '<', '=', '>',
 '?', '@', '[', '\\\\', ']', '^', '_', '{', '|', '}', '~']

```

In [70]:

```

1 ### Get rid of stop words
2 preprocessed = []
3 for sent in word_tokenized:
4     for token in sent:
5         token = token.lower()
6         if token not in stop_words:
7             preprocessed.append(token)
8 print(preprocessed)

```

```
[ 'mds', 'master', "'s", 'program', 'ubc', 'british', 'columbia', 'mds',
'teaching', 'team', 'truly', 'multicultural', 'dr.', 'george', 'ph.d.',
'scotland', 'dr.', 'timbers', 'dr.', 'ostblom', 'dr.', 'rodríguez-areli
s', 'dr.', 'kolhatkar', 'canada', 'dr.', 'gelbart', 'phd', 'u.s' ]
```

Lemmatization

- For many NLP tasks (e.g., web search) we want to ignore morphological differences between words
 - Example: If your search term is "studying for ML quiz" you might want to include pages containing "tips to study for an ML quiz" or "here is how I studied for my ML quiz"
- Lemmatization converts inflected forms into the base form.

In [71]:

```

1 import nltk
2
3 nltk.download("wordnet")
4 nltk.download('omw-1.4')
5

```

```
[nltk_data] Downloading package wordnet to /Users/mathias/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /Users/mathias/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Out[71]: True

In [72]:

```

1 # nltk has a lemmatizer
2 from nltk.stem import WordNetLemmatizer
3
4 lemmatizer = WordNetLemmatizer()
5 print("Lemma of studying: ", lemmatizer.lemmatize("studying", "v"))
6 print("Lemma of studied: ", lemmatizer.lemmatize("studied", "v"))

```

```
Lemma of studying: study
Lemma of studied: study
```

Stemming

- Has a similar purpose but it is a crude chopping of affixes
 - automates, automatic, automation* all reduced to *automat*.
- Usually these reduced forms (stems) are not actual words themselves.
- A popular stemming algorithm for English is PorterStemmer.

- Beware that it can be aggressive sometimes.

In [73]:

```

1 from nltk.stem.porter import PorterStemmer
2
3 text = (
4     "UBC is located in the beautiful province of British Columbia... "
5     "It's very close to the U.S. border."
6 )
7 ps = PorterStemmer()
8 tokenized = word_tokenize(text)
9 stemmed = [ps.stem(token) for token in tokenized]
10 print("Before stemming: ", text)
11 print("\n\nAfter stemming: ", " ".join(stemmed))

```

Before stemming: UBC is located in the beautiful province of British Columbia... It's very close to the U.S. border.

After stemming: ubc is locat in the beauti provinc of british columbia ... it 's veri close to the u.s. border .

Other tools for preprocessing

- We used [Natural Language Processing Toolkit \(nltk\)](https://www.nltk.org/) (<https://www.nltk.org/>) above
- Many available tools
- [spaCy](https://spacy.io/) (<https://spacy.io/>)

[spaCy](https://spacy.io/) (<https://spacy.io/>)

- Industrial strength NLP library.
- Lightweight, fast, and convenient to use.
- spaCy does many things that we did above in one line of code!
- Also has [multi-lingual](https://spacy.io/models/xx) (<https://spacy.io/models/xx>) support.

In [74]:

```

1 import spacy
2
3 # Load the model
4 nlp = spacy.load("en_core_web_md")
5 text = (
6     "MDS is a Master's program at UBC in British Columbia. "
7     "MDS teaching team is truly multicultural!! "
8     "Dr. George did his Ph.D. in Scotland. "
9     "Dr. Timbers, Dr. Ostblom, Dr. Rodríguez-Arelis, and Dr. Kolhatkar
10    "Dr. Gelbart did his PhD in the U.S."
11 )
12
13 doc = nlp(text)

```

In [75]:

```

1 # Accessing tokens
2 tokens = [token for token in doc]
3 print("\nTokens: ", tokens)
4
5 # Accessing lemma
6 lemmas = [token.lemma_ for token in doc]
7 print("\nLemmas: ", lemmas)
8
9 # Accessing pos
10 pos = [token.pos_ for token in doc]
11 print("\nPOS: ", pos)

```

Tokens: [MDS, is, a, Master, 's, program, at, UBC, in, British, Columbi
a, ., MDS, teaching, team, is, truly, multicultural, !, !, Dr., George, d
id, his, Ph.D., in, Scotland, ., Dr., Timbers, , Dr., Ostblom, , Dr., R
odríguez, -, Arelis, , and, Dr., Kolhatkar, did, theirs, in, Canada, .,
Dr., Gelbart, did, his, PhD, in, the, U.S.]

Lemmas: ['mds', 'be', 'a', 'Master', "'s", 'program', 'at', 'UBC', 'in',
'British', 'Columbia', '.', 'mds', 'teaching', 'team', 'be', 'truly', 'mu
lticultural', '!', '!', 'Dr.', 'George', 'do', 'his', 'ph.d.', 'in', 'Sco
tland', '.', 'Dr.', 'Timbers', ',', 'Dr.', 'Ostblom', ',', 'Dr.', 'Rodríg
uez', '-', 'Arelis', ',', 'and', 'Dr.', 'Kolhatkar', 'do', 'theirs', 'i
n', 'Canada', '.', 'Dr.', 'Gelbart', 'do', 'his', 'phd', 'in', 'the', 'U.
S.']

POS: ['NOUN', 'AUX', 'DET', 'PROPN', 'PART', 'NOUN', 'ADP', 'PROPN', 'AD
P', 'PROPN', 'PROPN', 'PUNCT', 'NOUN', 'NOUN', 'AUX', 'ADV', 'AD
J', 'PUNCT', 'PUNCT', 'PROPN', 'PROPN', 'VERB', 'PRON', 'NOUN', 'ADP', 'P
ROPN', 'PUNCT', 'PROPN', 'PUNCT', 'PROPN', 'PROPN', 'PUNCT', 'PR
OPN', 'PROPN', 'PUNCT', 'PROPN', 'PUNCT', 'CCONJ', 'PROPN', 'PROPN', 'VER
B', 'PRON', 'ADP', 'PROPN', 'PUNCT', 'PROPN', 'PROPN', 'VERB', 'PRON', 'N
OUN', 'ADP', 'DET', 'PROPN']

Other typical NLP tasks

In order to understand text, we usually are interested in extracting information from text. Some common tasks in NLP pipeline are:

- Part of speech tagging
 - Assigning part-of-speech tags to all words in a sentence.
- Named entity recognition
 - Labelling named “real-world” objects, like persons, companies or locations.
- Coreference resolution
 - Deciding whether two strings (e.g., UBC vs University of British Columbia) refer to the same entity
- Dependency parsing
 - Representing grammatical structure of a sentence

Extracting named-entities using spaCy

In [78]:

```

1 from spacy import displacy
2 import warnings
3 warnings.filterwarnings("ignore", category=DeprecationWarning)
4
5 doc = nlp(
6     "University of British Columbia "
7     "is located in the beautiful "
8     "province of British Columbia."
9 )
10 displacy.render(doc, style="ent")
11 # Text and label of named entity span
12 print("Named entities:\n", [(ent.text, ent.label_) for ent in doc.ents])
13 print("\nORG means: ", spacy.explain("ORG"))
14 print("GPE means: ", spacy.explain("GPE"))

```

University of British Columbia **ORG** is located in the beautiful province of **British Columbia**
GPE.

Named entities:

```
[('University of British Columbia', 'ORG'), ('British Columbia', 'GPE')]
```

ORG means: Companies, agencies, institutions, etc.

GPE means: Countries, cities, states

Dependency parsing using spaCy

In [79]:

```

1 doc = nlp("I like cats")
2 displacy.render(doc, style="dep")

```

I PRON like VERB cats NOUN nsubj dobj

Many other things possible

- A powerful tool
- All my Capstone groups last year used this tool.
- You can build your own rule-based searches.
- You can also access word vectors using spaCy with bigger models. (Currently we are using `en_core_web_md` model.)

CPSC 330

Applied Machine Learning

Lecture 18: Multi-class classification and introduction to computer vision

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

```
In [2]: 1 %load_ext autoreload  
2 %autoreload 2
```

```
In [3]: 1 import glob  
2 import os  
3 import sys  
4 sys.path.append("../code/.")  
5  
6 from mglearn_utils import discrete_scatter, cm3, plot_2d_classification  
7 import matplotlib.pyplot as plt  
8 import numpy as np  
9 import pandas as pd  
10 from sklearn import datasets  
11 from sklearn.dummy import DummyClassifier  
12 from sklearn.ensemble import RandomForestClassifier  
13 from sklearn.linear_model import LinearRegression, LogisticRegression  
14 from sklearn.metrics import (  
15     classification_report,  
16     confusion_matrix,  
17     plot_confusion_matrix,  
18 )  
19 from sklearn.model_selection import train_test_split  
20 from sklearn.pipeline import Pipeline, make_pipeline  
21 from sklearn.svm import SVC
```

Learning objectives

- Apply classifiers to multi-class classification algorithms.
- Explain the role of neural networks in machine learning, and the pros/cons of using them.
- Explain why the methods we've learned previously would not be effective on image data.

Typesetting math: 100% • Apply pre-trained neural networks to classification and regression problems.

- Utilize pre-trained networks as feature extractors and combine them with models we've learned previously.

Multi-class, meta-strategies

- So far we have been talking about binary classification
- Can we use these classifiers when there are more than two classes?
 - ["ImageNet" computer vision competition \(<http://www.image-net.org/challenges/LSVRC/>\),](http://www.image-net.org/challenges/LSVRC/) for example, has 1000 classes
- Can we use decision trees or KNNs for multi-class classification?
- What about logistic regression and Linear SVMs?

- Many linear classification models don't extend naturally to the multiclass case.
- A common technique is to reduce multiclass classification into several instances of binary classification problems.
- Two kind of "hacky" ways to reduce multi-class classification into binary classification:
 - the one-vs.-rest approach
 - the one-vs.-one approach

One vs. Rest

- $1v\{2,3\}$, $2v\{1,3\}$, $3v\{1,2\}$
- Learn a binary model for each class which tries to separate that class from all of the other classes.
- If you have k classes, it'll train k binary classifiers, one for each class.
- Trained on imbalanced datasets containing all examples.
- Given a test point, get scores from all binary classifiers (e.g., raw scores for logistic regression).

- The classifier which has the highest score for this class "wins" and that's going to be the prediction for this class.
- Since we have one binary classifier per class, for each class, we have coefficients per feature and an intercept.

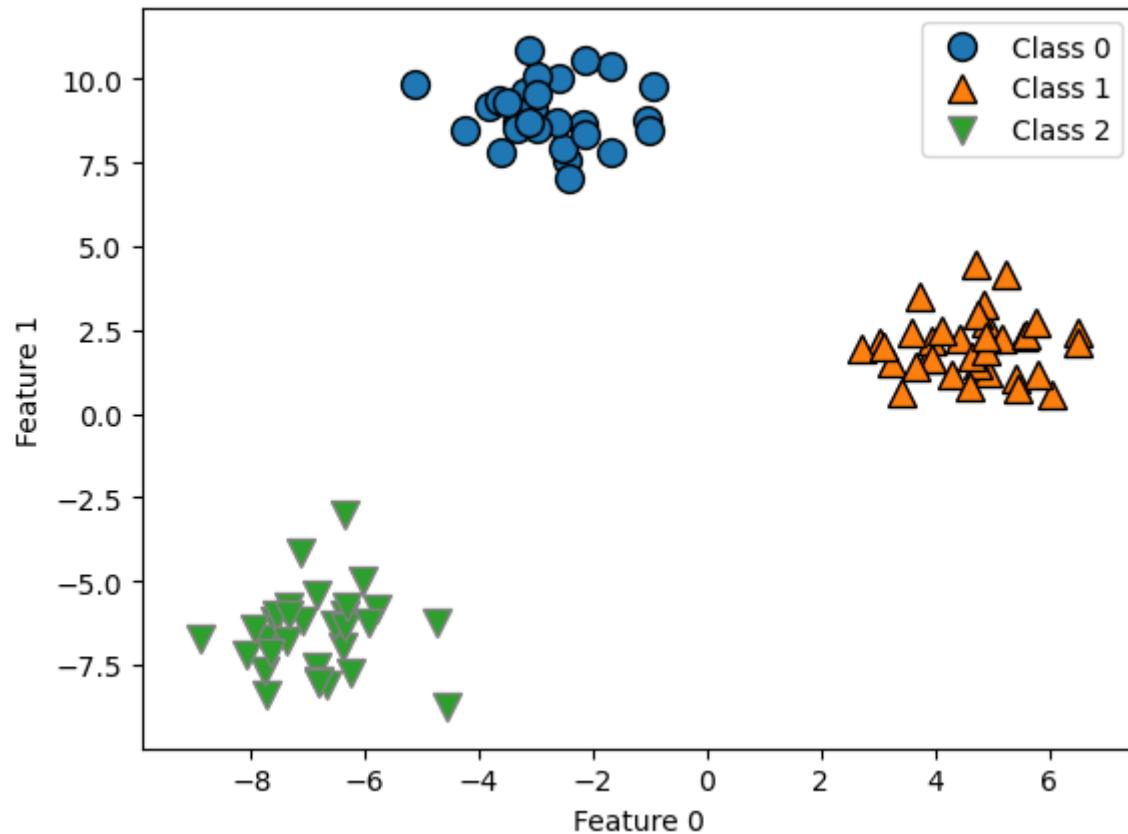
Let's create some synthetic data with two features and three classes.

In [4]:

```

1 from sklearn.datasets import make_blobs
2
3 X, y = make_blobs(centers=3, n_samples=120, random_state=42)
4 X_train, X_test, y_train, y_test = train_test_split(
5     X, y, test_size=0.2, random_state=123
6 )
7 discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
8 plt.xlabel("Feature 0")
9 plt.ylabel("Feature 1")
10 plt.legend(["Class 0", "Class 1", "Class 2"]);

```



In [5]:

```

1 lr = LogisticRegression(max_iter=2000, multi_class="ovr")
2 lr.fit(X_train, y_train)
3 print("Coefficient shape: ", lr.coef_.shape)
4 print("Intercept shape: ", lr.intercept_.shape)
5 lr.coef_

```

Coefficient shape: (3, 2)

Intercept shape: (3,)

Out[5]: array([[-0.65123329, 1.0536117],
 [1.35418019, -0.28647501],
 [-0.63320669, -0.72513556]])

- This learns three binary linear models.
- So we have coefficients for two features for each of these three linear models.
- Also we have three intercepts, one for each class.

Typesetting math: 100%

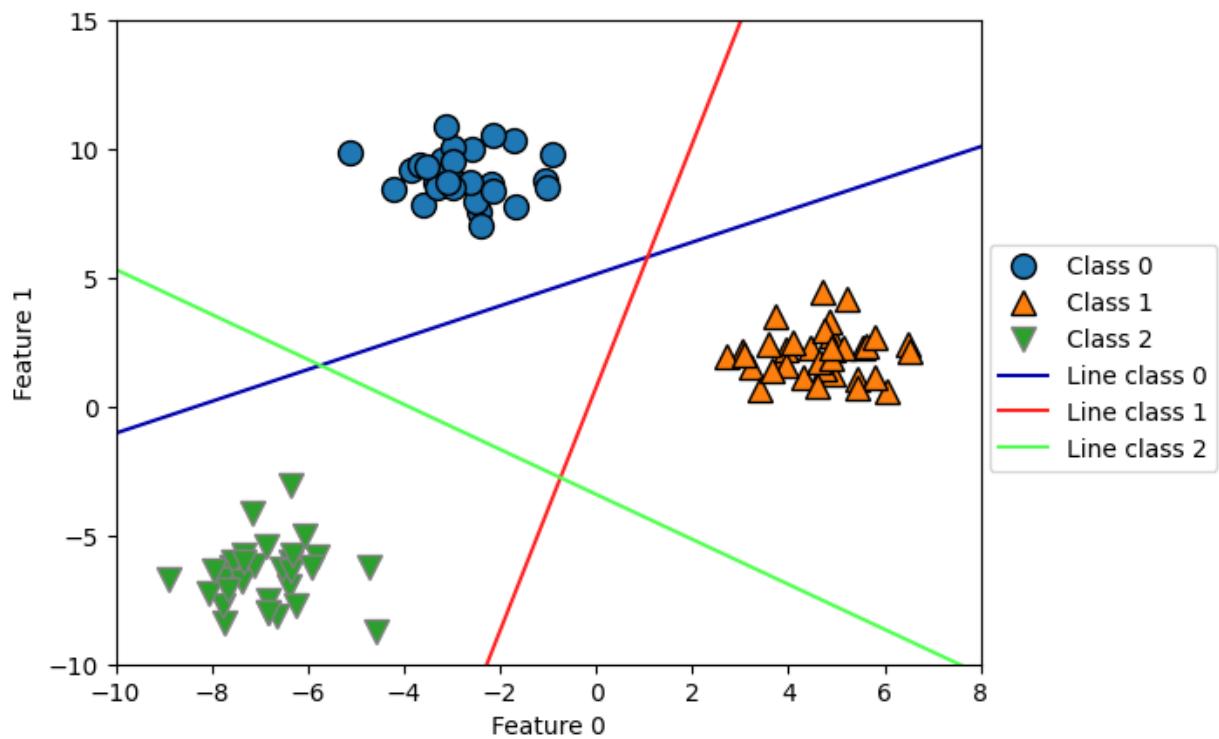
[Code credit \(<https://learning.oreilly.com/library/view/introduction-to-machine/9781449369880/ch02.html#linear-models>\)](https://learning.oreilly.com/library/view/introduction-to-machine/9781449369880/ch02.html#linear-models)

In [6]:

```

1 discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
2 line = np.linspace(-15, 15)
3 for coef, intercept, color in zip(lr.coef_, lr.intercept_, cm3.colors):
4     plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
5 plt.ylim(-10, 15)
6 plt.xlim(-10, 8)
7 plt.xlabel("Feature 0")
8 plt.ylabel("Feature 1")
9 plt.legend(
10     ["Class 0", "Class 1", "Class 2", "Line class 0", "Line class 1", "Line class 2"],
11     loc=(1.01, 0.3),
12 );

```



In [7]:

```

1 def plot_test_points():
2     test_points = [[-4.0, 12], [-2, 0.0], [-8, 3.0], [4, 8.5]]
3     plt.plot(test_points[0][0], test_points[0][1], "k*", markersize=16)
4     plt.plot(test_points[1][0], test_points[1][1], "k*", markersize=16)
5     plt.plot(test_points[2][0], test_points[2][1], "k*", markersize=16)
6     plt.plot(test_points[3][0], test_points[3][1], "k*", markersize=16)

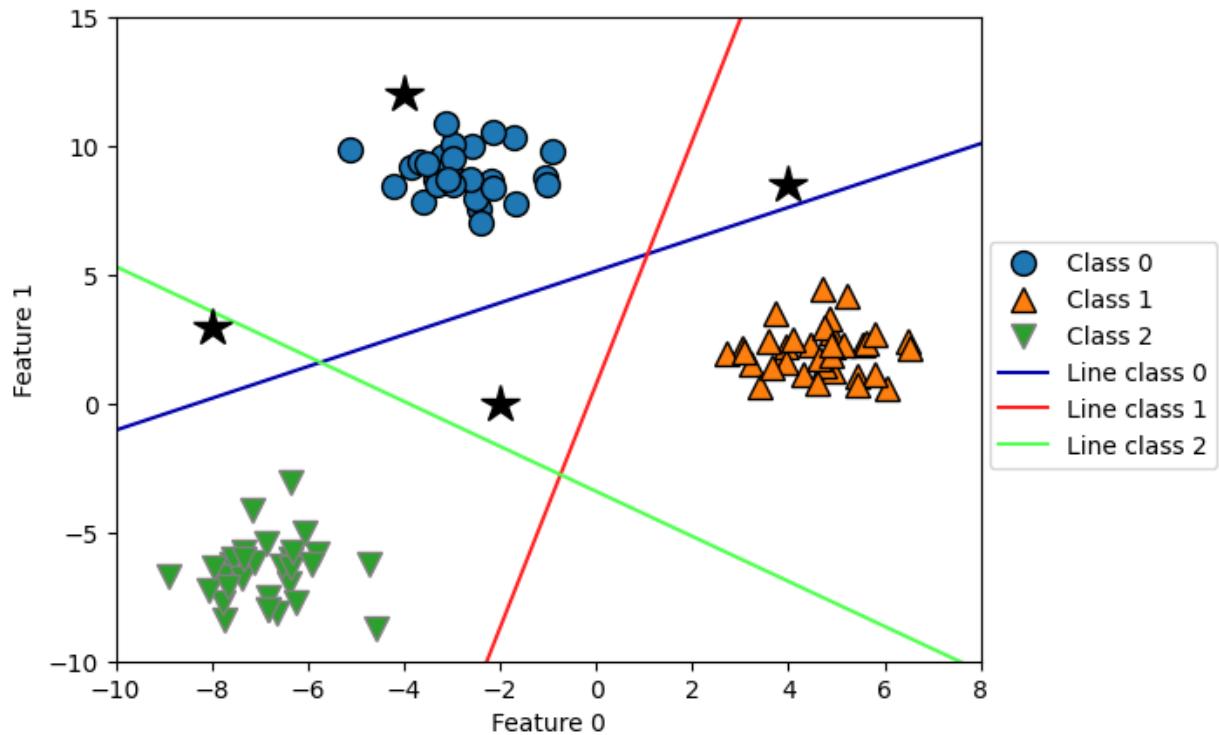
```

- How would you classify the following points?
 - Pick the class with the highest value for the classification formula.

Typesetting math: 100%

In [8]:

```
1 # You don't have to understand the code.
2 discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
3 line = np.linspace(-15, 15)
4 for coef, intercept, color in zip(lr.coef_, lr.intercept_, cm3.colors):
5     plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
6 plot_test_points()
7 plt.ylim(-10, 15)
8 plt.xlim(-10, 8)
9 plt.xlabel("Feature 0")
10 plt.ylabel("Feature 1")
11 plt.legend(
12     ["Class 0", "Class 1", "Class 2", "Line class 0", "Line class 1", "Line class 2"],
13     loc=(1.01, 0.3),
14 );
```



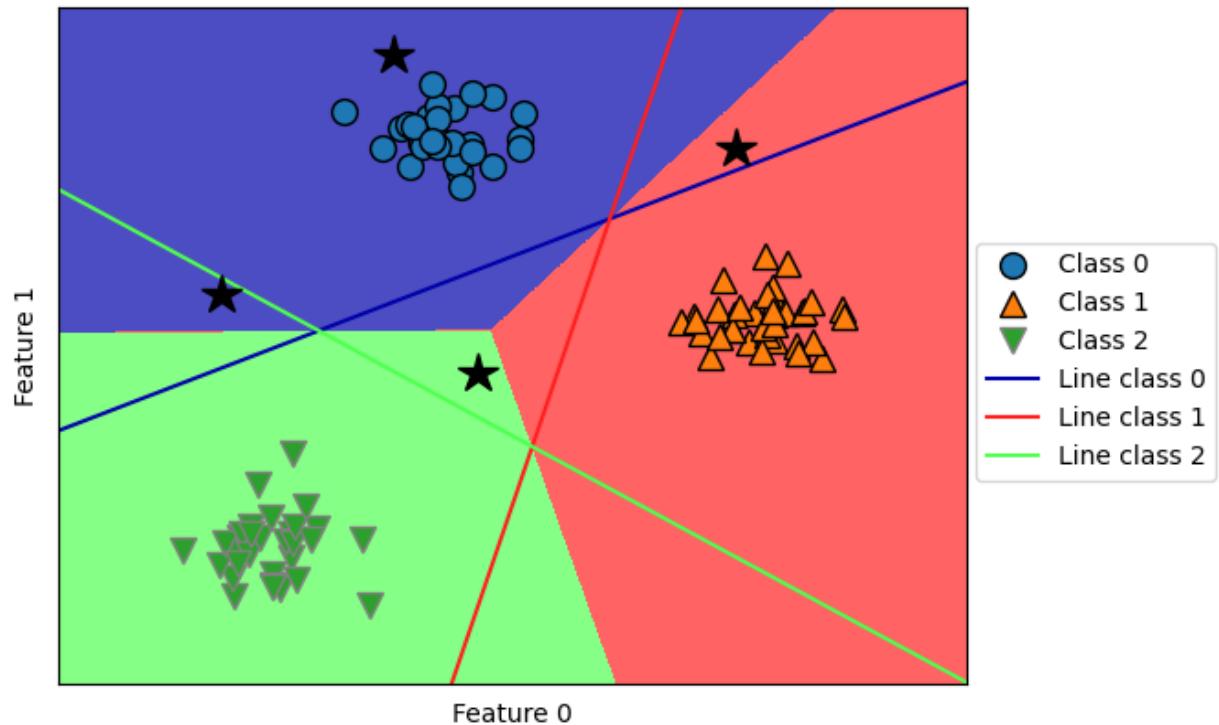
Typesetting math: 100%

In [9]:

```

1 # You don't have to understand the code below.
2 plot_2d_classification(lr, X_train, fill=True, alpha=0.7)
3 discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
4 line = np.linspace(-15, 15)
5 for coef, intercept, color in zip(lr.coef_, lr.intercept_, cm3.colors):
6     plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
7 plot_test_points()
8 plt.legend(
9     ["Class 0", "Class 1", "Class 2", "Line class 0", "Line class 1", "Line class 2"],
10    loc=(1.01, 0.3),
11 )
12 plt.xlabel("Feature 0")
13 plt.ylabel("Feature 1");

```



In [10]:

```

1 test_points = [[-4.0, 12], [-2, 0.0], [-8, 3.0], [4, 8.5]]
2 lr.predict_proba(test_points)

```

Out[10]: array([[9.99645770e-01, 1.77111249e-04, 1.77118671e-04],
 [4.92621189e-02, 2.36353992e-01, 7.14383889e-01],
 [6.11446337e-01, 6.67363501e-06, 3.88546990e-01],
 [4.26997745e-01, 5.72993816e-01, 8.43813466e-06]])

In [11]:

```
1 lr.predict(test_points)
```

Out[11]: array([0, 2, 0, 1])

One Vs. One approach

- Build a binary model for each pair of classes.
- 1v2, 1v3, 2v3

Typesetting math: 100% Trains $\frac{n \times (n-1)}{2}$ binary classifiers

- Trained on relatively balanced subsets

One Vs. One prediction

- Apply all of the classifiers on the test example.
- Count how often each class was predicted.
- Predict the class with most votes.

Using OVR and OVO as wrappers

- You can use these strategies as meta-strategies for any binary classifiers.
 - [OneVsRestClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html) (<https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html>)
 - [OneVsOneClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsOneClassifier.html) (<https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsOneClassifier.html>)

- When do we use `OneVsRestClassifier` and `OneVsOneClassifier`
- It's not that likely for you to need `OneVsRestClassifier` or `OneVsOneClassifier` because most of the methods you'll use will have native multi-class support.
- However, it's good to know in case you ever need to extend a binary classifier (perhaps one you've implemented on your own).

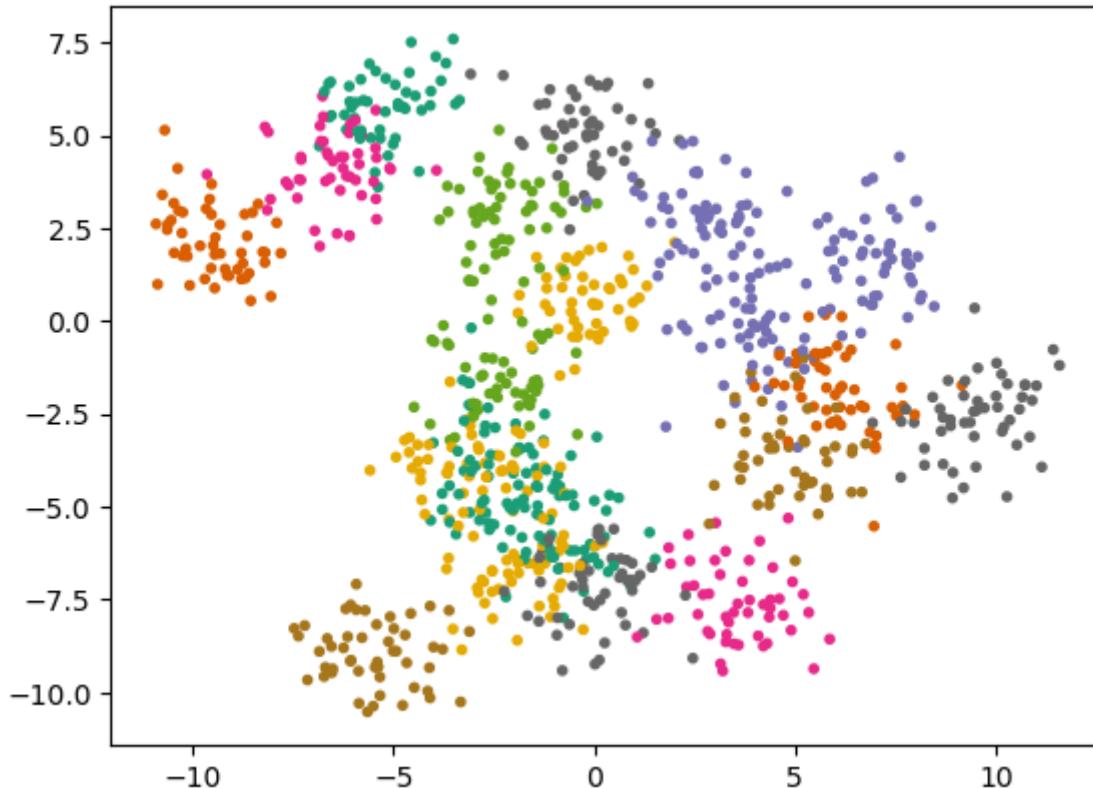
In [12]: 1 `from sklearn.multiclass import OneVsOneClassifier, OneVsRestClassifier`

In [13]:

```

1 # Let's examine the time taken by OneVsRestClassifier and OneVsOneClass
2
3 # generate blobs with fixed random generator
4 X_multi, y_multi = make_blobs(n_samples=1000, centers=20, random_state=
5
6 X_train_multi, X_test_multi, y_train_multi, y_test_multi = train_test_s
7     X_multi, y_multi
8 )
9
10 plt.scatter(*X_multi.T, c=y_multi, marker=". ", cmap="Dark2");

```



In [14]:

```

1 model = OneVsOneClassifier(LogisticRegression())
2 %timeit model.fit(X_train_multi, y_train_multi);
3 print("With OVO wrapper")
4 print(model.score(X_train_multi, y_train_multi))
5 print(model.score(X_test_multi, y_test_multi))

```

209 ms ± 8.73 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
 With OVO wrapper
 0.8066666666666666
 0.756

Typesetting math: 100%

In [15]:

```
1 model = OneVsRestClassifier(LogisticRegression())
2 %timeit model.fit(X_train_multi, y_train_multi);
3 print("With OVR wrapper")
4 print(model.score(X_train_multi, y_train_multi))
5 print(model.score(X_test_multi, y_test_multi))
```

29.2 ms ± 1.49 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

With OVR wrapper

0.7546666666666667

0.668

- As expected OVO takes more time compared to OVR.
- [Here \(https://scikit-learn.org/stable/modules/multiclass.html\)](https://scikit-learn.org/stable/modules/multiclass.html) you will find summary of how scikit-learn handles multi-class classification for different classifiers.

True/False

1. One-vs.-one strategy uses all the available data when training each binary classifier.
2. For a 100-class classification problem, one-vs.-rest multi-class strategy will create 100 binary classifiers.

Multi-class classification on [HappyDB](#) (<https://www.kaggle.com/ritresearch/happydb>) corpus

Let's examine precision, recall, and f1-score of different classes in the [HappyDB](#) (<https://www.kaggle.com/ritresearch/happydb>) corpus.

Typesetting math: 100%

```
In [16]: 1 df = pd.read_csv("../data/cleaned_hm.csv", index_col=0)
2 sample_df = df.dropna()
3 sample_df.head()
4 sample_df = sample_df.rename(
5     columns={"cleaned_hm": "moment", "ground_truth_category": "target"}
6 )
7 sample_df.head()
```

Out[16]:

	wid	reflection_period	original_hm	moment	modified	num_sentence	target	predicte
	hmid							
27676	206	24h	We had a serious talk with some friends of our...	We had a serious talk with some friends of our...	True	2	bonding	
27678	45	24h	I meditated last night.	I meditated last night.	True	1	leisure	
27697	498	24h	My grandmother start to walk from the bed afte...	My grandmother start to walk from the bed afte...	True	1	affection	
27705	5732	24h	I picked my daughter up from the airport and w...	I picked my daughter up from the airport and w...	True	1	bonding	
27715	2272	24h	when i received flowers from my best friend	when i received flowers from my best friend	True	1	bonding	

```
In [17]: 1 sample_df["target"].value_counts()
```

Out[17]:

affection	4810
achievement	4276
bonding	1750
enjoy_the_moment	1514
leisure	1306
nature	252
exercise	217

Name: target, dtype: int64

It's a multiclass classification problem!

```
In [18]: 1 train_df, test_df = train_test_split(sample_df, test_size=0.3, random_s
2 X_train_happy, y_train_happy = train_df["moment"], train_df["target"]
3 X_test_happy, y_test_happy = test_df["moment"], test_df["target"]
```

In [19]:

```

1 from sklearn.feature_extraction.text import CountVectorizer
2
3 pipe_lr = make_pipeline(
4     CountVectorizer(stop_words="english"), LogisticRegression(max_iter=200)
5 )

```

In [20]:

```
1 pipe_lr.fit(X_train_happy, y_train_happy)
```

Out[20]: Pipeline(steps=[('countvectorizer', CountVectorizer(stop_words='english')), ('logisticregression', LogisticRegression(max_iter=200))])

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [21]:

```

1 preds = pipe_lr.predict(X_test_happy)[:5]
2 preds

```

Out[21]: array(['achievement', 'affection', 'bonding', 'enjoy_the_moment', 'affection'], dtype=object)

Note that the output of `predict_proba` now contains a probability for each class:

In [22]:

```
1 pipe_lr.predict_proba(X_test_happy)[:5]
```

Out[22]: array([[7.06472042e-01, 3.74986136e-02, 5.65462935e-02, 4.48416453e-02, 3.05695812e-02, 1.10293654e-01, 1.37781706e-02], [4.51114062e-03, 9.89340017e-01, 5.83647210e-04, 3.70764136e-03, 2.90911962e-04, 6.37035303e-04, 9.29606301e-04], [2.13299803e-03, 1.51563896e-02, 9.78842547e-01, 1.36505813e-03, 1.26043214e-03, 9.41950993e-04, 3.00623854e-04], [1.13093070e-01, 9.17955178e-02, 2.38830890e-02, 5.06663413e-01, 7.31006757e-03, 2.49573509e-01, 7.68133360e-03], [7.72068573e-02, 5.53170559e-01, 3.87164821e-02, 8.14773263e-02, 2.38206330e-02, 2.08447747e-01, 1.71603952e-02]])

And you'll see that each row adds up to 1, as expected:

In [23]:

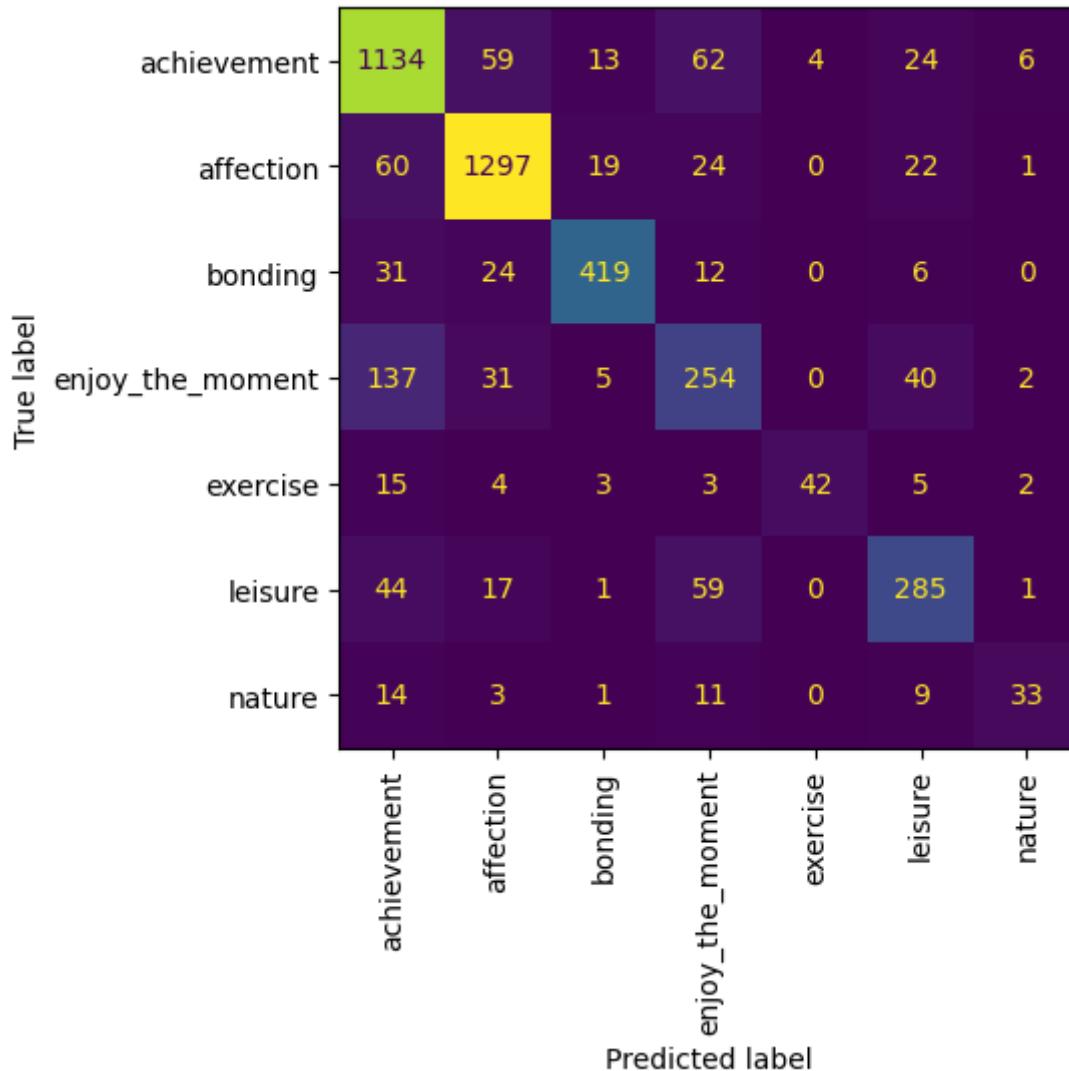
```
1 pipe_lr.predict_proba(X_test_happy).sum(axis=1)
```

Out[23]: array([1., 1., 1., ..., 1., 1., 1.])

We can also make a confusion matrix:

In [24]:

```
1 from sklearn.metrics import plot_confusion_matrix, ConfusionMatrixDisplay
2
3 ConfusionMatrixDisplay.from_estimator(pipe_lr,
4     X_test_happy,
5     y_test_happy,
6     values_format="d",
7     xticks_rotation="vertical",
8     colorbar=False,
9 );
```



And print the classification report.

```
In [25]: 1 print(classification_report(y_test_happy, pipe_lr.predict(X_test_happy))
```

	precision	recall	f1-score	support
achievement	0.79	0.87	0.83	1302
affection	0.90	0.91	0.91	1423
bonding	0.91	0.85	0.88	492
enjoy_the_moment	0.60	0.54	0.57	469
exercise	0.91	0.57	0.70	74
leisure	0.73	0.70	0.71	407
nature	0.73	0.46	0.57	71
accuracy			0.82	4238
macro avg	0.80	0.70	0.74	4238
weighted avg	0.82	0.82	0.81	4238

- Seems like there is a lot of variation in the scores for different classes.
- The model is performing pretty well on *affection* class but not that well on *enjoy_the_moment* and *nature* classes.

How are the predictions made?

```
In [26]: 1 pipe_lr.predict_proba(X_test_happy)[:5]
```

```
Out[26]: array([[7.06472042e-01, 3.74986136e-02, 5.65462935e-02, 4.48416453e-02,
       3.05695812e-02, 1.10293654e-01, 1.37781706e-02],
      [4.51114062e-03, 9.89340017e-01, 5.83647210e-04, 3.70764136e-03,
       2.90911962e-04, 6.37035303e-04, 9.29606301e-04],
      [2.13299803e-03, 1.51563896e-02, 9.78842547e-01, 1.36505813e-03,
       1.26043214e-03, 9.41950993e-04, 3.00623854e-04],
      [1.13093070e-01, 9.17955178e-02, 2.38830890e-02, 5.06663413e-01,
       7.31006757e-03, 2.49573509e-01, 7.68133360e-03],
      [7.72068573e-02, 5.53170559e-01, 3.87164821e-02, 8.14773263e-02,
       2.38206330e-02, 2.08447747e-01, 1.71603952e-02]])
```

```
In [27]: 1 np.argmax(pipe_lr.predict_proba(X_test_happy), axis=1)
```

```
Out[27]: array([0, 1, 2, ..., 1, 0, 2])
```

```
In [28]: 1 classes = pipe_lr.classes_
2 classes
```

```
Out[28]: array(['achievement', 'affection', 'bonding', 'enjoy_the_moment',
   'exercise', 'leisure', 'nature'], dtype=object)
```

```
In [29]: 1 y_hat = pipe_lr.predict(X_test_happy)
2 y_hat
```

```
Out[29]: array(['achievement', 'affection', 'bonding', ..., 'affection',
   'achievement', 'bonding'], dtype=object)
```

Typesetting math: 100%

How many coefficients have we learned?

In [30]: 1 pipe_lr.named_steps["logisticregression"].coef_.shape

Out[30]: (7, 8060)

- We have one coefficient per feature *per class*.
- Let's examine them.

In [31]:

```
1 feature_names = pipe_lr.named_steps["countvectorizer"].get_feature_name
2 lr_coefs = pd.DataFrame(
3     data=pipe_lr.named_steps["logisticregression"].coef_.T,
4     index=feature_names,
5     columns=classes,
6 ).sort_values("bonding", ascending=False)
7 lr_coefs
```

Out[31]:

	achievement	affection	bonding	enjoy_the_moment	exercise	leisure	nature
friend	-1.687528	-0.183608	5.589866	-1.707829	0.330456	-1.769253	-0.572105
friends	-1.304287	0.052779	5.246125	-1.992716	0.328815	-1.559656	-0.771061
roommate	-1.327202	-0.690686	3.418085	-1.138203	-0.078646	-0.070284	-0.113063
coworkers	-0.588496	-0.606175	3.011513	-1.098932	-0.088536	-0.518752	-0.110623
coworker	-0.934857	-0.591312	2.770930	-0.560366	-0.093246	-0.415496	-0.175652
...
feelings	0.057296	1.194994	-0.898329	-0.123828	-0.103999	-0.093033	-0.033103
jogging	-0.046343	-0.319887	-0.909238	-0.098763	1.521341	-0.130390	-0.016719
telling	-0.436909	0.870929	-1.070986	0.694175	-0.066820	0.038809	-0.029197
drive	-0.150838	0.584506	-1.184986	0.891987	-0.239585	-0.453318	0.552235
boy	1.398748	0.288127	-1.327434	0.138895	-0.261471	-0.162182	-0.074683

8060 rows × 7 columns

The interpretation is a feature importance for predicting a certain class. For example:

In [32]: 1 lr_coefs.loc["friend"][2]

Out[32]: 5.589866397597406

- This means that if the value for the feature "friend" is bigger, you are more likely to predict class "bonding".

- If you want a general feature importance irrespective of class, you could try looking at the sum of the squares of the coefficients, which is what sklearn does:

Typesetting math: 100%

```
In [33]: 1 (lr_coefs ** 2).sum(axis=1).sort_values(ascending=False)
```

```
Out[33]: friend      4.061151e+01
friends     3.633188e+01
husband     2.764409e+01
wife        2.542235e+01
son         2.361098e+01
...
curiosity   1.996211e-11
gang        1.728753e-11
teases       1.728753e-11
horror       1.728753e-11
cinemas     1.728753e-11
Length: 8060, dtype: float64
```

```
In [34]: 1 ?LogisticRegression
```

```
Init signature:
LogisticRegression(
    penalty='l2',
    *,
    dual=False,
    tol=0.0001,
    C=1.0,
    fit_intercept=True,
    intercept_scaling=1,
    class_weight=None,
    random_state=None,
    solver='lbfgs',
    max_iter=100,
    multi_class='auto',
    verbose=0,
    warm_start=False,
    n_jobs=None,
    l1_ratio=None,
)
```

- We can see that there's a `multi_class` parameter, that can be set to '`ovr`' or '`multinomial`', or you can have it automatically choose between the two, which is the default.
 - In CPSC 340 we discuss in detail the difference between these two approaches.
 - In CPSC 340 we make an argument for preferring '`multinomial`', but in short it doesn't matter which one you choose.

Break (5 min)



Intro to computer vision

- [Computer vision \(\[https://en.wikipedia.org/wiki/Computer_vision\]\(https://en.wikipedia.org/wiki/Computer_vision\)\)](https://en.wikipedia.org/wiki/Computer_vision) refers to understanding images/videos, usually using ML/AI.
- Computer vision has many tasks of interest:
 - image classification: is this a cat or a dog?
 - object localization: where are the people in this image?
 - image segmentation: what are the various parts of this image?
 - motion detection: what moved between frames of a video?
 - and much more...
- We will focus on image classification.

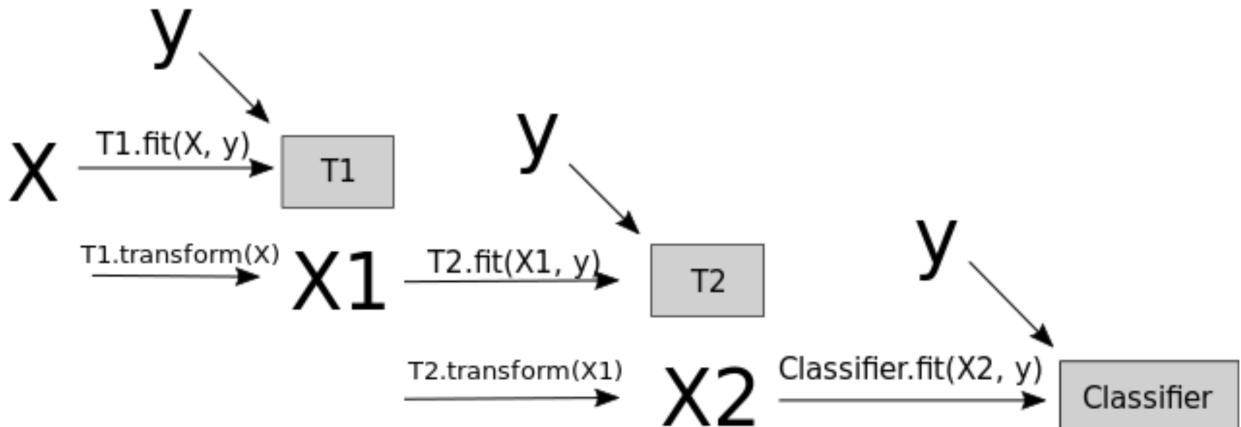
Intro to neural networks

- Very popular these days under the name **deep learning**.
- Neural networks apply a sequence of transformations on your input data.
- At a very high level you can think of them as `Pipelines` in `sklearn`.
- A neural network is a model that's sort of like its own pipeline
 - It involves a series of transformations ("layers") internally.
 - The output is the prediction.

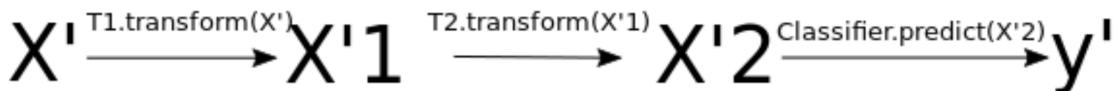
Typeetting math: 100%



`pipe.fit(X, y)`



`pipe.predict(X')`



[Source \(<https://amueller.github.io/COMS4995-s20/slides/aml-04-preprocessing/#18>\)](https://amueller.github.io/COMS4995-s20/slides/aml-04-preprocessing/#18)

- They can be viewed as a generalization of linear models where we apply a series of transformations.
- Here is graphical representation of logistic regression model.
 - We have 4 features: $x[0], x[1], x[2], x[3]$

In [35]:

```
1 from mglearn.utils import plot_logistic_regression_graph, plot_single_h
2
3 display(plot_logistic_regression_graph())
```

<graphviz.graphs.Digraph at 0x196ce9b70>

- Below we are adding one "layer" of transformations in between features and the target.
- We are repeating the process of computing the weighted sum multiple times.
- The **hidden units** (e.g., $h[1], h[2], \dots$) represent the intermediate processing steps.

In [36]:

```
1 display(plot_single_hidden_layer_graph())
```

<graphviz.graphs.Digraph at 0x196cea650>

- Now we are adding one more layer of transformations.

Typesetting math: 100%

In [37]:

```
1 display(plot_two_hidden_layer_graph())
```

```
<graphviz.graphs.Digraph at 0x196dc1540>
```

- Important question: how many features before/after transformation.
 - e.g. scaling doesn't change the number of features
 - OHE increases the number of features
- With a neural net, you specify the number of features after each transformation.
 - In the above, it goes from 4 to 3 to 3 to 1 (output).

- To make them really powerful compared to the linear models, we apply a non-linear function to the weighted sum for each hidden node.

Terminology

- Neural network = neural net
- Deep learning ~ using neural networks

Why neural networks?

- They can learn very complex functions.
 - The fundamental tradeoff is primarily controlled by the **number of layers** and **layer sizes**.
 - More layers / bigger layers --> more complex model.
 - You can generally get a model that will not underfit.

Why neural networks?

- They work really well for structured data:
 - 1D sequence, e.g. timeseries, language
 - 2D image
 - 3D image or video
- They've had some incredible successes in the last 10 years.
- Transfer learning (coming later today) is really useful.

Why not neural networks?

- Often they require a lot of data.
- They require a lot of compute time, and, to be faster, specialized hardware called **GPUs** (https://en.wikipedia.org/wiki/Graphics_processing_unit).
- They have huge numbers of hyperparameters and are a huge pain to tune.
 - Think of each layer having hyperparameters, plus some overall hyperparameters.
 - Being slow compounds this problem.
- They are not interpretable.

Typesetting math: 100%

Why not neural networks?

- When you call `fit`, you are not guaranteed to get the optimal.
 - There are now a bunch of hyperparameters specific to `fit`, rather than the model.
 - You never really know if `fit` was successful or not.
 - You never really know if you should have run `fit` for longer.
- I don't recommend training them on your own without further training
 - Take CPSC 340 and other courses if you're interested.
 - I'll show you some ways to use neural networks without calling `fit`.

Deep learning software

- scikit-learn has [MLPRegressor \(\[https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html\]\(https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html\)\)](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html) and [MLPClassifier \(\[https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html\]\(https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html\)\)](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html) but they aren't very flexible.
 - In general you'll want to leave the scikit-learn ecosystem when using neural networks.
 - Fun fact: these classes were contributed to scikit-learn by a UBC graduate student.
- There's been a lot of deep learning software out there.

- The current big players are:
 1. [TensorFlow \(<https://www.tensorflow.org>\)](https://www.tensorflow.org)
 2. [PyTorch \(<http://pytorch.org>\)](http://pytorch.org)
- Both are heavily used in industry.
- If interested, see [comparison of deep learning software \(\[https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software\]\(https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software\)\)](https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software).

Neural networks on image data

In [38]:

```
1 import matplotlib as mpl
2 from sklearn.datasets import fetch_lfw_people
3
4 mpl.rcParams.update(mpl.rcParamsDefault)
5 plt.rcParams["image.cmap"] = "gray"
```

Typesetting math: 100%

In [39]:

```

1 from sklearn.datasets import fetch_lfw_people
2
3 people = fetch_lfw_people(min_faces_per_person=40, resize=0.7)
4
5 fig, axes = plt.subplots(2, 5, figsize=(12, 6), subplot_kw={"xticks": ()}
6 for target, image, ax in zip(people.target, people.images, axes.ravel())
7     ax.imshow(image)
8     ax.set_title(people.target_names[target])
9
10 plt.show();

```



In [40]:

```

1 image_shape = people.images[0].shape
2 print("people.images.shape: {}".format(people.images.shape))
3 print("Number of classes: {}".format(len(people.target_names)))

```

people.images.shape: (1867, 87, 65)
Number of classes: 19

There are 1,867 images stored as arrays of 5655 pixels (87 by 65), of 19 different people:

In [41]:

```

1 # count how often each target appears
2 counts = np.bincount(people.target)
3 df = pd.DataFrame(counts, columns=["count"], index=people.target_names)
4 df.sort_values("count", ascending=False)

```

Out[41]:

	count
George W Bush	530
Colin Powell	236
Tony Blair	144
Donald Rumsfeld	121
Gerhard Schroeder	109
Ariel Sharon	77
Hugo Chavez	71
Junichiro Koizumi	60
Jean Chretien	55
John Ashcroft	53
Jacques Chirac	52
Serena Williams	52
Vladimir Putin	49
Luiz Inacio Lula da Silva	48
Gloria Macapagal Arroyo	44
Arnold Schwarzenegger	42
Jennifer Capriati	42
Laura Bush	41
Lleyton Hewitt	41

Let's make the data less skewed by taking only 20 images of the each person.

In [42]:

```

1 mask = np.zeros(people.target.shape, dtype=bool)
2 for target in np.unique(people.target):
3     mask[np.where(people.target == target)[0][:20]] = 1
4
5 X_people = people.data[mask]
6 y_people = people.target[mask]

```

In [43]:

```
1 X_people.shape
```

Out[43]: (380, 5655)

Typesetting math: 100%

```
In [44]: 1 # scale the grayscale values to be between 0 and 1
          2 # instead of 0 and 255 for better numeric stability
          3 x_people = x_people / 255.0
```

```
In [45]: 1 x_train, x_test, y_train, y_test = train_test_split(
          2     x_people, y_people, random_state=123
          3 )
```

```
In [46]: 1 x_train
```

```
Out[46]: array([[0.0000000e+00, 0.0000000e+00, 0.0000000e+00, ... , 0.0000000e+00,
                 0.0000000e+00, 0.0000000e+00],
                [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, ... , 0.0000000e+00,
                 0.0000000e+00, 0.0000000e+00],
                [5.1262336e-06, 0.0000000e+00, 0.0000000e+00, ... , 0.0000000e+00,
                 0.0000000e+00, 0.0000000e+00],
                ... ,
                [0.0000000e+00, 5.1262336e-06, 1.0252467e-05, ... , 0.0000000e+00,
                 0.0000000e+00, 0.0000000e+00],
                [0.0000000e+00, 0.0000000e+00, 0.0000000e+00, ... , 9.1246958e-04,
                 7.2279893e-04, 7.1254646e-04],
                [1.2354223e-03, 1.2354223e-03, 1.2405486e-03, ... , 1.5840061e-03,
                 1.5532487e-03, 1.5532487e-03]], dtype=float32)
```

Now the data is in this tabular format that we are used to. Now we can use our usual classification methods.

```
In [47]: 1 lr = LogisticRegression(max_iter=4000)
```

```
In [48]: 1 lr.fit(x_train, y_train);
```

```
In [49]: 1 lr.score(x_train, y_train)
```

```
Out[49]: 0.07017543859649122
```

```
In [50]: 1 lr.score(x_test, y_test)
```

```
Out[50]: 0.010526315789473684
```

We are getting very poor test results :(.

- Why flattening images is a bad idea?
 - By "flattening" the image we throw away useful information.
- What the computer sees:

```
In [51]: 1 list(X_train[0])[150:200]
```

```
Out[51]: [1.0252467e-05,
 1.0252467e-05,
 5.1262336e-06,
 1.0252467e-05,
 5.1262336e-06,
 5.1262336e-06,
 5.1262336e-06,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 5.638857e-05,
 5.638857e-05,
 0.0005895169,
 0.0007227989,
 0.00095860567,
 0.0017788031,
 0.0017531719,
 0.0022657954,
 0.0027015249,
 0.0027681661,
 0.003142381,
 0.0031526333,
 0.0031372549,
 0.0031577598,
 0.0031526333,
 0.0030859925,
 0.0030654876,
 0.0030706136,
 0.0030193515,
 0.0030039728,
 0.0029937204,
 0.0030603614,
 0.0021376396,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0]
```

- Hard to classify this!

Typesetting math: 100% [Convolutional neural networks \(\[https://en.wikipedia.org/wiki/Convolutional_neural_network\]\(https://en.wikipedia.org/wiki/Convolutional_neural_network\)\)](https://en.wikipedia.org/wiki/Convolutional_neural_network)
(CNNs) can take in images without flattening them.

- We won't cover CNNs here, but they are in CPSC 340.

Transfer learning

- In practice, very few people train an entire CNN from scratch because it requires a large dataset, powerful computers, and a huge amount of human effort to train the model.
- Instead, a common practice is to download a pre-trained model and fine tune it for your task.
- This is called **transfer learning**.
- Transfer learning is one of the most common techniques used in the context of computer vision and natural language processing.
 - In the last lecture we used pre-trained embeddings to train create text representation.

Using pre-trained models out-of-the-box

Recall this example I showed you in the intro video (our very first lecture).

In [52]:

```

1 def classify_image(img, topn=4):
2     clf = vgg16(weights=VGG16_Weights.DEFAULT)    # Loading the pre-train
3     preprocess = transforms.Compose(
4         [
5             transforms.Resize(299),
6             transforms.CenterCrop(299),
7             transforms.ToTensor(),
8             transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229
9         ]
10     )    # Defining a preprocessor to transform a given image so that it'
11
12     with open("../data/imagenet_classes.txt") as f:
13         classes = [line.strip() for line in f.readlines()]
14
15     img_t = preprocess(img)
16     batch_t = torch.unsqueeze(img_t, 0)
17     clf.eval()
18     output = clf(batch_t)
19     _, indices = torch.sort(output, descending=True)
20     probabilities = torch.nn.functional.softmax(output, dim=1)
21     d = {
22         "Class": [classes[idx] for idx in indices[0][:topn]],
23         "Probability score": [
24             np.round(probabilities[0, idx].item(), 3) for idx in indice
25         ],
26     }
27     df = pd.DataFrame(d, columns=["Class", "Probability score"])
28     return df

```

In [53]:

```

1 import torch
2 from PIL import Image
3 from torchvision import transforms
4 from torchvision.models import vgg16, VGG16_Weights, DenseNet121_Weight

```

Typesetting math: 100%

In [54]:

```
1 # Predict labels with associated probabilities for unseen images
2 images = glob.glob("../data/test_images/*.*")
3 for image in images:
4     img = Image.open(image)
5     img.load()
6     plt.imshow(img)
7     plt.show()
8     df = classify_image(img)
9     print(df.to_string(index=False))
10    print("-----")
```



- We got these predictions without "doing the ML ourselves".
- We are using **pre-trained** vgg16 model which is available in `torchvision`
- `torchvision` has many such pre-trained models available that have been very successful across a wide range of tasks: AlexNet, VGG, ResNet, Inception, MobileNet, etc.
- Many of these models have been pre-trained on famous datasets like **ImageNet**.

ImageNet

- [ImageNet \(http://www.image-net.org/\)](http://www.image-net.org/) is an image dataset that became a very popular benchmark in the field ~10 years ago.
- [Wikipedia article \(https://en.wikipedia.org/wiki/ImageNet\)](https://en.wikipedia.org/wiki/ImageNet)
- There are 14 million images and 1000 classes.
- Here are some example classes.

Typesetting math: 100%

```
In [55]: 1 with open("../data/imagenet_classes.txt") as f:
2     classes = [line.strip() for line in f.readlines()]
3 classes[100:110]
```

```
Out[55]: ['black swan, Cygnus atratus',
'tusker',
'echidna, spiny anteater, anteater',
'platypus, duckbill, duckbilled platypus, duck-billed platypus, Ornithorhynchus anatinus',
'wallaby, brush kangaroo',
'koala, koala bear, kangaroo bear, native bear, Phascolarctos cinereus',
'wombat',
'jellyfish',
'sea anemone, anemone',
'brain coral']
```

Let's see what labels this pre-trained model give us for CPSC 330 teaching team.

```
In [56]: 1 # Predict labels with associated probabilities for unseen images
2 images = glob.glob("../data/_330_teaching_team/*.*")
3 for image in images:
4     img = Image.open(image)
5     img.load()
6     plt.imshow(img)
7     plt.show()
8     df = classify_image(img)
9     print(df.to_string(index=False))
10    print("-----")
```



- It's not doing very well here because ImageNet don't have classes for Giulia, Michelle, Amir, or Mathias.
- Here we are using pre-trained models out-of-the-box.
- Can we use pre-trained models for our own classification problem with our classes?

Typesetting math: 100%

Using pre-trained models as feature extractor

- Here we will use pre-trained models to extract features.
- We will pass our specific data through a pre-trained network to get a feature vector for each example in the data.
- You train a machine learning classifier such as logistic regression or random forest using these extracted feature vectors.

Typesetting math: 100%

In [57]:

```
1 # Attribution: [Code from PyTorch docs](https://pytorch.org/tutorials/b
2
3 import copy
4 import os
5 import time
6
7 import matplotlib.pyplot as plt
8 import numpy as np
9 import torch
10 import torch.nn as nn
11 import torch.optim as optim
12 import torchvision
13 from torch.optim import lr_scheduler
14 from torchvision import datasets, models, transforms
15
16 data_transforms = {
17     "train": transforms.Compose(
18         [
19             transforms.RandomResizedCrop(224),
20             transforms.RandomHorizontalFlip(),
21             transforms.ToTensor(),
22             transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
23         ])
24     ),
25     "val": transforms.Compose(
26         [
27             transforms.Resize(256),
28             transforms.CenterCrop(224),
29             transforms.ToTensor(),
30             transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
31         ])
32     ),
33 }
34 data_dir = "../data/hymenoptera_data"
35 image_datasets = {
36     x: datasets.ImageFolder(os.path.join(data_dir, x), data_transforms[
37         for x in ["train", "val"]
38     ])
39     dataloaders = {
40         x: torch.utils.data.DataLoader(
41             image_datasets[x], batch_size=4, shuffle=True, num_workers=4
42         )
43         for x in ["train", "val"]
44     }
45     dataset_sizes = {x: len(image_datasets[x]) for x in ["train", "val"]}
46     class_names = image_datasets["train"].classes
47
48     device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

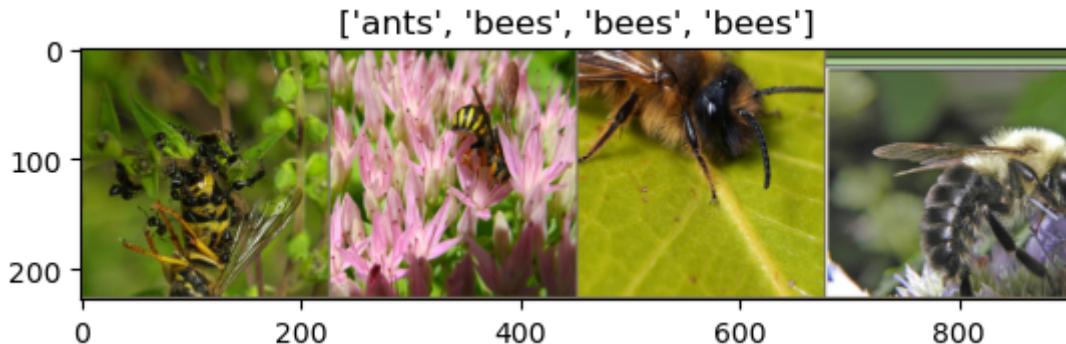
Typesetting math: 100%

In [58]:

```

1 def imshow(inp, title=None):
2     """Imshow for Tensor."""
3     inp = inp.numpy().transpose((1, 2, 0))
4     mean = np.array([0.485, 0.456, 0.406])
5     std = np.array([0.229, 0.224, 0.225])
6     inp = std * inp + mean
7     inp = np.clip(inp, 0, 1)
8     plt.imshow(inp)
9     if title is not None:
10         plt.title(title)
11     plt.pause(0.001) # pause a bit so that plots are updated
12
13
14 # Get a batch of training data
15 inputs, classes = next(iter(dataloaders["train"]))
16
17 # Make a grid from batch
18 out = torchvision.utils.make_grid(inputs)
19
20 imshow(out, title=[class_names[x] for x in classes])

```



In [59]:

```

1 print(f"Classes: {image_datasets['train'].classes}")
2 print(
3     f"Class count: {image_datasets['train'].targets.count(0)}, {image_d
4 )
5 print(f"Samples: {len(image_datasets['train'])}")
6 print(f"First sample: {image_datasets['train'].samples[0]}")

```

Classes: ['ants', 'bees']
 Class count: 123, 121
 Samples: 244
 First sample: ('../data/hymenoptera_data/train/ants/0013035.jpg', 0)

Typesetting math: 100%

```
In [60]: 1 def get_features(model, train_loader, valid_loader):
2     """Extract output of squeezeNet model"""
3
4     with torch.no_grad(): # turn off computational graph stuff
5         Z_train = torch.empty((0, 1024)) # Initialize empty tensors
6         y_train = torch.empty((0))
7         Z_valid = torch.empty((0, 1024))
8         y_valid = torch.empty((0))
9         for X, y in train_loader:
10             Z_train = torch.cat((Z_train, model(X)), dim=0)
11             y_train = torch.cat((y_train, y))
12         for X, y in valid_loader:
13             Z_valid = torch.cat((Z_valid, model(X)), dim=0)
14             y_valid = torch.cat((y_valid, y))
15     return Z_train.detach(), y_train.detach(), Z_valid.detach(), y_valid
```

```
In [61]: 1 densenet = models.densenet121(weights=DenseNet121_Weights.DEFAULT)
2 densenet.classifier = nn.Identity() # remove that last "classification"
```

```
In [ ]: 1 Z_train, y_train, Z_valid, y_valid = get_features(
2     densenet, dataloaders["train"], dataloaders["val"]
3 )
```

Now we have some extracted features.

```
In [ ]: 1 Z_train.shape
```

```
In [ ]: 1 from sklearn.pipeline import Pipeline, make_pipeline
2 from sklearn.preprocessing import StandardScaler
3
4 pipe = make_pipeline(StandardScaler(), LogisticRegression(max_iter=2000)
5 pipe.fit(Z_train, y_train)
6 pipe.score(Z_train, y_train)
```

```
In [ ]: 1 pipe.score(Z_valid, y_valid)
```

- This is great accuracy for so little data (We only have 244 examples.) and little effort!!!

TODO

- Compare this to accuracy with flattened images and logistic regression
- Try this out with the Faces dataset.

Random cool stuff

- Style transfer: given a "content image" and a "style image", create a new image with the

Typesetting math: 100%

Content of one and the style of the other.

- Here is the [original paper from 2015 \(<https://arxiv.org/pdf/1508.06576.pdf>\)](https://arxiv.org/pdf/1508.06576.pdf), see Figure 2.

- Here are more in [this 2016 paper](https://arxiv.org/pdf/1601.04589.pdf) (<https://arxiv.org/pdf/1601.04589.pdf>); see, e.g. Figures 1 and 7.
- This has been done for video as well; see [this video from 2016](https://www.youtube.com/watch?v=Khuj4ASldmU) (<https://www.youtube.com/watch?v=Khuj4ASldmU>).
- [Image captioning](https://cs.stanford.edu/people/karpathy/sfmltalk.pdf) (<https://cs.stanford.edu/people/karpathy/sfmltalk.pdf>): Transfer learning with NLP and vision
- Colourization: see [this 2016 project](http://iizuka.cs.tsukuba.ac.jp/projects/colorization/en/) (<http://iizuka.cs.tsukuba.ac.jp/projects/colorization/en/>).
- Inceptionism: let the neural network "make things up"
 - [2015 article](https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html) (<https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>)
 - "Deep dream" [video from 2015](https://www.youtube.com/watch?v=dbQh1I_uvjo) (https://www.youtube.com/watch?v=dbQh1I_uvjo).

Summary

- Multi-class classification refers to classification with >2 classes.
 - Most sklearn classifiers work out of the box.
 - With LogisticRegression the situation with the coefficients is a bit funky, we get 1 coefficient per feature per class.
- Flattening images throws away a lot of useful information (sort of like one-hot encoding on ordinal variable!).
- Neural networks are a flexible class of models.
 - They are hard to train - a lot more on that in CPSC 340.
 - They generally require leaving the sklearn ecosystem to tensorflow or pytorch.
 - They are particularly powerful for structured input like images, videos, audio, etc.
- The good news is we can use pre-trained neural networks.
 - This saves us a huge amount of time/cost/effort/resources.
 - We can use these pre-trained networks directly or use them as feature transformers.
- My general recommendation: don't use deep learning unless there is good reason to.

Lecture 18: Time series

UBC 2022-23

Instructor: Mathias Lécuyer

Announcements

- HW8 released, due on April 12.

```
In [1]: 1 %load_ext autoreload  
2 %autoreload 2
```

```
In [2]: 1 import sys  
2 import matplotlib.pyplot as plt  
3 import numpy as np  
4 import pandas as pd  
5 from sklearn.compose import ColumnTransformer, make_column_transformer  
6 from sklearn.dummy import DummyClassifier  
7 from sklearn.ensemble import RandomForestClassifier  
8 from sklearn.impute import SimpleImputer  
9 from sklearn.linear_model import LogisticRegression  
10 from sklearn.model_selection import (  
11     TimeSeriesSplit,  
12     cross_val_score,  
13     cross_validate,  
14     train_test_split,  
15 )  
16 from sklearn.pipeline import Pipeline, make_pipeline  
17 from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler  
18  
19 sys.path.append("../code/.")  
20  
21 plt.rcParams["font.size"] = 16  
22 from datetime import datetime
```

Learning objectives

- Explain the pitfalls of train/test splitting with time series data.
- Appropriately split time series data, both train/test split and cross-validation.
- Perform time series feature engineering:
 - Encode time as various features in a tabular dataset
 - Create lag-based features
- Explain how can you forecast multiple time steps into the future.
- Explain the challenges of time series data with unequally spaced time points.

Motivation

- **Time series** is a collection of data points indexed in time order.
- Time series is everywhere:
 - Physical sciences (e.g., weather forecasting)
 - Economics, finance (e.g., stocks, market trends)
 - Engineering (e.g., energy consumption)
 - Social sciences
 - Sports analytics

Let's start with a simple example from [Introduction to Machine Learning with Python book](#) (<https://learning.oreilly.com/library/view/introduction-to-machine/9781449369880/>).

In New York city there is a network of bike rental stations with a subscription system. The stations are all around the city. The anonymized data is available [here](#) (<https://ride.citibikenyc.com/system-data>).

We will focus on the task of predicting how many people will rent a bicycle from a particular station for a given time and day. We might be interested in knowing this so that we know whether there will be any bikes left at the station for a particular day and time.

```
In [3]: 1 def load_citibike(file):
2     data_mine = pd.read_csv(file)
3     data_mine['one'] = 1
4     data_mine['starttime'] = pd.to_datetime(data_mine.starttime)
5     data_starttime = data_mine.set_index("starttime")
6     data_resampled = data_starttime.resample("3h").sum(numeric_only=True)
7
8     return data_resampled.one
9
10 citibike = load_citibike("../data/201508-citibike-tripdata.csv")
11 citibike.head()
```

```
Out[3]: starttime
2015-08-01 00:00:00    1135
2015-08-01 03:00:00     302
2015-08-01 06:00:00    1781
2015-08-01 09:00:00    7126
2015-08-01 12:00:00    8442
Freq: 3H, Name: one, dtype: int64
```

- The only feature we have is the date time feature.
 - Example: 2015-08-01 00:00:00
- The target is the number of rentals in the next 3 hours.
 - Example: 1135 rentals between 2015-08-01 00:00:00 and 2015-08-01 02:59:59

```
In [4]: 1 citibike.index.min()
```

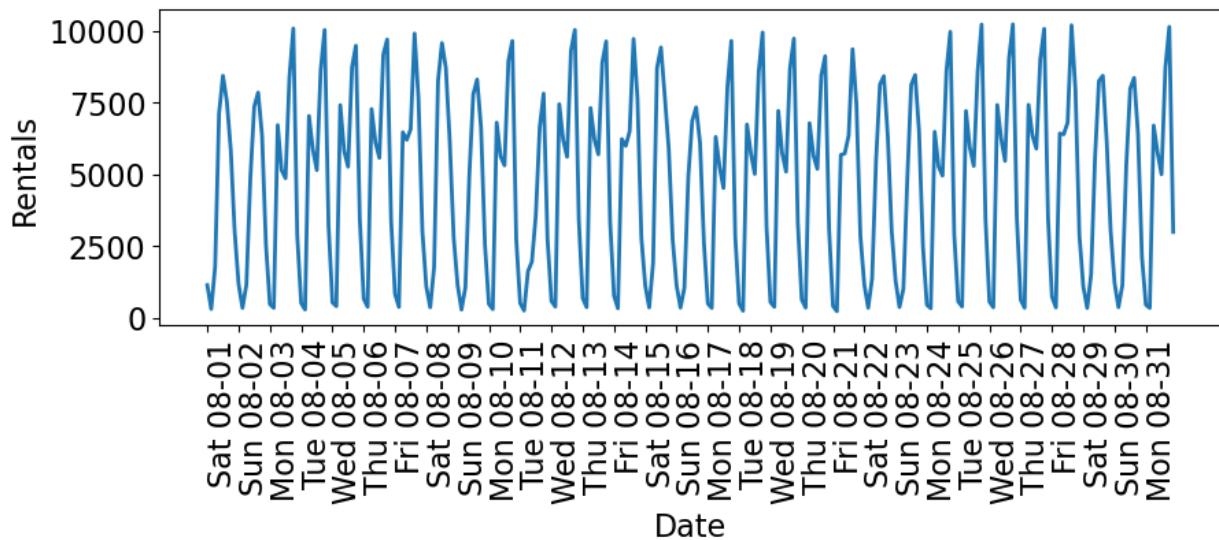
Out[4]: Timestamp('2015-08-01 00:00:00', freq='3H')

```
In [5]: 1 citibike.index.max()
```

Out[5]: Timestamp('2015-08-31 21:00:00', freq='3H')

We have data for August 2015.

```
In [6]: 1 plt.figure(figsize=(10, 3))
2 xticks = pd.date_range(start=citibike.index.min(), end=citibike.index.m
3 plt.xticks(xticks, xticks.strftime("%a %m-%d"), rotation=90, ha="left")
4 plt.plot(citibike, linewidth=2)
5 plt.xlabel("Date")
6 plt.ylabel("Rentals");
```



- We see the day and night pattern
- We see the weekend and weekday pattern

- Questions you might want to answer: How many people are likely to rent a bike at this station tomorrow at 3pm given everything we know about rentals in the past?
- We want to learn from the past and predict the future.

Train/test split for temporal data

- What will happen if we split this data the usual way?

```
In [7]: 1 train_df, test_df = train_test_split(citibike, test_size=0.2, random_st
```

```
In [8]: 1 test_df.head()
```

```
Out[8]: starttime
2015-08-26 12:00:00    5469
2015-08-12 09:00:00    6199
2015-08-19 03:00:00     373
2015-08-07 12:00:00   6573
2015-08-03 09:00:00   5144
Name: one, dtype: int64
```

```
In [9]: 1 train_df.index.max()
```

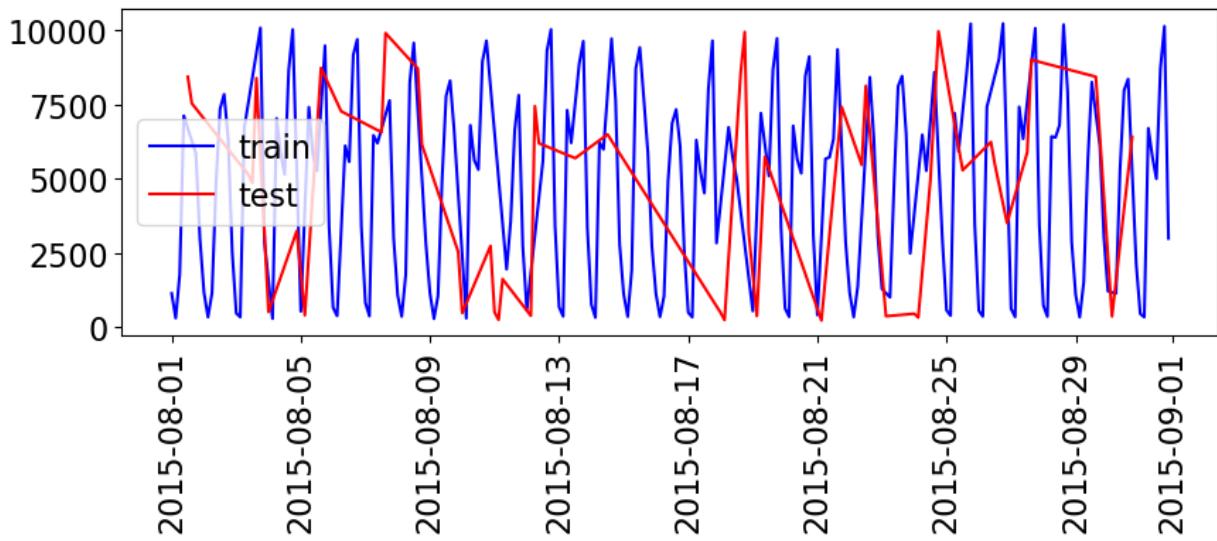
```
Out[9]: Timestamp('2015-08-31 21:00:00')
```

```
In [10]: 1 test_df.index.min()
```

```
Out[10]: Timestamp('2015-08-01 12:00:00')
```

- So, we are training on data that came after our test data!
- If we want to forecast, **we aren't allowed to know what happened in the future!**
- There may be cases where this is OK, e.g. if you aren't trying to forecast and just want to understand your data (maybe you're not even splitting).
- But, for our purposes, we want to avoid this.

```
In [11]: 1 plt.figure(figsize=(10, 3))
2 train_df_sort = train_df.sort_index()
3 test_df_sort = test_df.sort_index()
4
5 plt.plot(train_df_sort, "b", label="train")
6 plt.plot(test_df_sort, "r", label="test")
7 plt.xticks(rotation="vertical")
8 plt.legend();
```



We'll split the data as follows:

- We have total 248 data points.

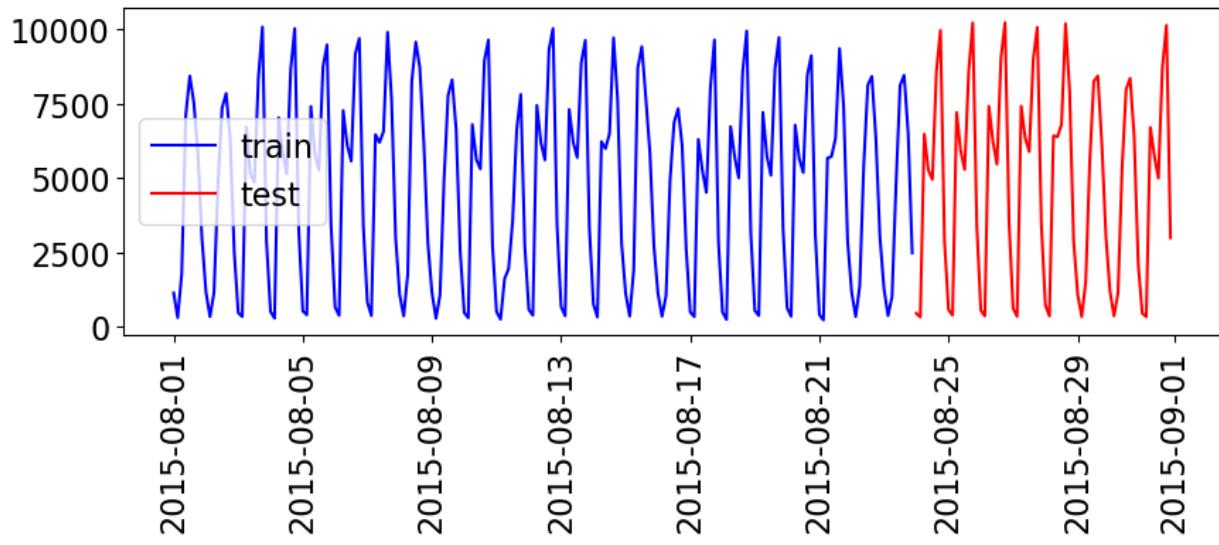
- We'll use the first 184 data points corresponding to the first 23 days as training data - And the remaining 64 data points corresponding to the remaining 8 days as test data.

In [12]: 1 citibike.shape

Out[12]: (248,)

In [13]: 1 n_train = 184
2 train_df = citibike[:184]
3 test_df = citibike[184:]

In [14]: 1 plt.figure(figsize=(10, 3))
2 train_df_sort = train_df.sort_index()
3 test_df_sort = test_df.sort_index()
4
5 plt.plot(train_df_sort, "b", label="train")
6 plt.plot(test_df_sort, "r", label="test")
7 plt.xticks(rotation="vertical")
8 plt.legend();



- This split is looking reasonable now.

Training models

- In this toy data, we just have a single feature: the date time feature.
- We need to encode this feature if we want to build machine learning models.
- A common way that dates are stored on computers is using POSIX time, which is the number of seconds since January 1970 00:00:00 (this is beginning of Unix time).
- Let's start with encoding this feature as a single integer representing this POSIX time.

```
In [15]: 1 X = (
2     citibike.index.astype("int64").values.reshape(-1, 1) // 10 ** 9
3 ) # convert to POSIX time by dividing by 10**9
4 y = citibike.values
```

```
In [16]: 1 y_train = train_df.values
2 y_test = test_df.values
3 # convert to POSIX time by dividing by 10**9
4 X_train = train_df.index.astype("int64").values.reshape(-1, 1) // 10 ** 9
5 X_test = test_df.index.astype("int64").values.reshape(-1, 1) // 10 ** 9
```

```
In [17]: 1 X_train[:10]
```

Out[17]: array([[1438387200],
 [1438398000],
 [1438408800],
 [1438419600],
 [1438430400],
 [1438441200],
 [1438452000],
 [1438462800],
 [1438473600],
 [1438484400]])

```
In [18]: 1 y_train[:10]
```

Out[18]: array([1135, 302, 1781, 7126, 8442, 7544, 5858, 2991, 1173, 336])

- Our prediction task is a regression task.

Let's try random forest regression.

```
In [19]: 1 from sklearn.ensemble import RandomForestRegressor
2
3 regressor = RandomForestRegressor(n_estimators=100, random_state=0)
4 regressor.fit(X_train, y_train)
5
6 print("Train-set R^2: {:.2f}".format(regressor.score(X_train, y_train)))
7 print("Test-set R^2: {:.2f}".format(regressor.score(X_test, y_test)))
```

Train-set R²: 0.94
 Test-set R²: -0.09

In [20]:

```

1 ## Code credit: https://learning.oreilly.com/library/view/introduction-
2
3
4 def eval_on_features(features, target, regressor):
5     # split the given features into a training and a test set
6     X_train, X_test = features[:n_train], features[n_train:]
7     # also split the target array
8     y_train, y_test = target[:n_train], target[n_train:]
9     regressor.fit(X_train, y_train)
10    print("Train-set R^2: {:.2f}".format(regressor.score(X_train, y_train)))
11    print("Test-set R^2: {:.2f}".format(regressor.score(X_test, y_test)))
12    y_pred = regressor.predict(X_test)
13    y_pred_train = regressor.predict(X_train)
14    plt.figure(figsize=(10, 3))
15
16    plt.xticks(range(0, len(X), 8), xticks.strftime("%a %m-%d"), rotation=45)
17
18    plt.plot(range(n_train), y_train, label="train")
19    plt.plot(range(n_train, len(y_test) + n_train), y_test, "--", label="test")
20    plt.plot(range(n_train), y_pred_train, "--", label="prediction train")
21
22    plt.plot(
23        range(n_train, len(y_test) + n_train), y_pred, "--", label="prediction test"
24    )
25    plt.legend(loc=(1.01, 0))
26    plt.xlabel("Date")
27    plt.ylabel("Rentals")

```

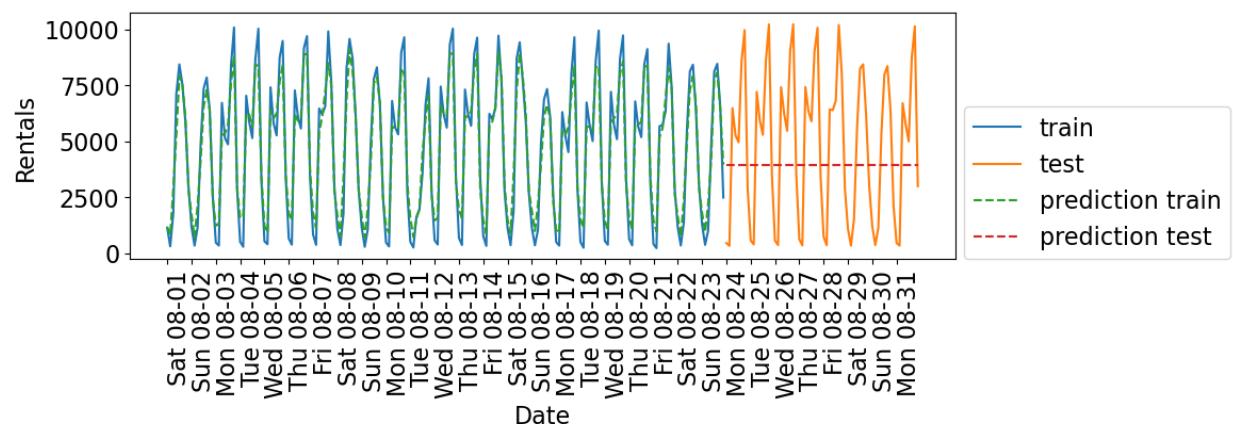
In [21]:

```

1 from sklearn.ensemble import RandomForestRegressor
2
3 regressor = RandomForestRegressor(n_estimators=100, random_state=0)
4 eval_on_features(X, y, regressor)

```

Train-set R²: 0.94
 Test-set R²: -0.09



- The predictions on the training set are pretty good
- But for the test data, a constant line is predicted ...
- What's going on?

- The model is based on only one feature: POSIX time feature.
- And the value of the POSIX time feature is outside the range of the feature values in the training set.
- Tree-based models cannot *extrapolate* to feature ranges outside the training data.
- The model predicted the target value of the closest point in the training set.

Can we come up with better features?

Feature engineering for date/time columns

- Note that our index is of this special type: `DatetimeIndex` (<https://pandas.pydata.org/docs/reference/api/pandas.DatetimeIndex.html>). We can extract all kinds of interesting information from it.

In [22]: 1 citibike.index

```
Out[22]: DatetimeIndex(['2015-08-01 00:00:00', '2015-08-01 03:00:00',
                       '2015-08-01 06:00:00', '2015-08-01 09:00:00',
                       '2015-08-01 12:00:00', '2015-08-01 15:00:00',
                       '2015-08-01 18:00:00', '2015-08-01 21:00:00',
                       '2015-08-02 00:00:00', '2015-08-02 03:00:00',
                       ...
                       '2015-08-30 18:00:00', '2015-08-30 21:00:00',
                       '2015-08-31 00:00:00', '2015-08-31 03:00:00',
                       '2015-08-31 06:00:00', '2015-08-31 09:00:00',
                       '2015-08-31 12:00:00', '2015-08-31 15:00:00',
                       '2015-08-31 18:00:00', '2015-08-31 21:00:00'],
                      dtype='datetime64[ns]', name='starttime', length=248, freq
                      ='3H')
```

In [23]: 1 citibike.index.month_name()

```
Out[23]: Index(['August', 'August', 'August', 'August', 'August', 'August', 'Augus
t',
               'August', 'August', 'August',
               ...
               'August', 'August', 'August', 'August', 'August', 'August', 'Augus
t',
               'August', 'August', 'August'],
              dtype='object', name='starttime', length=248)
```

In [24]: 1 citibike.index.dayofweek

```
Out[24]: Int64Index([5, 5, 5, 5, 5, 5, 5, 5, 6, 6,
                     ...
                     6, 6, 0, 0, 0, 0, 0, 0, 0, 0],
                     dtype='int64', name='starttime', length=248)
```

```
In [25]: 1 citibike.index.day_name()
```

```
Out[25]: Index(['Saturday', 'Saturday', 'Saturday', 'Saturday', 'Saturday', 'Saturday',
   'Saturday', 'Saturday', 'Sunday', 'Sunday',
   ...
   'Sunday', 'Sunday', 'Monday', 'Monday', 'Monday', 'Monday', 'Monday',
   'Monday', 'Monday'],
  dtype='object', name='starttime', length=248)
```

```
In [26]: 1 citibike.index.hour
```

```
Out[26]: Int64Index([ 0,  3,  6,  9, 12, 15, 18, 21,  0,  3,
   ...
   18, 21,  0,  3,  6,  9, 12, 15, 18, 21],
  dtype='int64', name='starttime', length=248)
```

- We noted before that the time of the day and day of the week seem quite important.
- Let's add these two features.

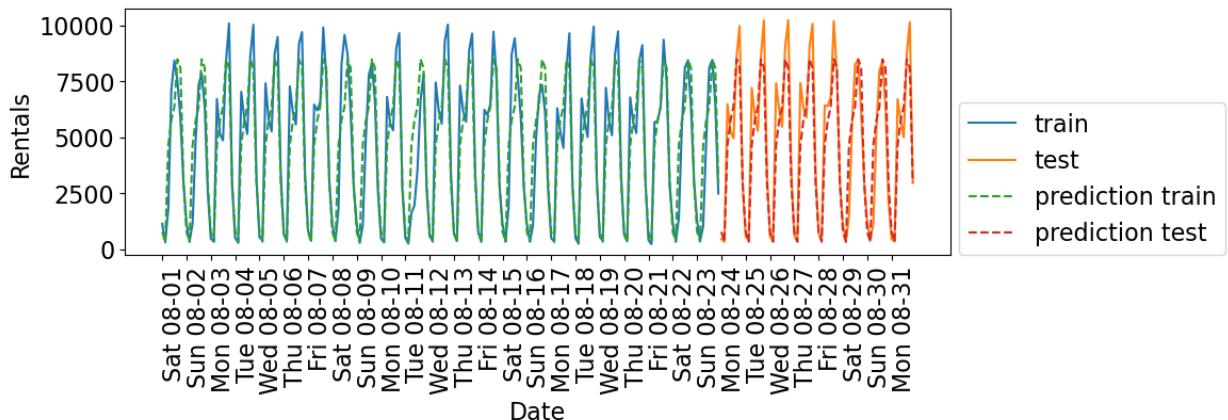
Let's first add the time of the day.

```
In [27]: 1 X_hour = citibike.index.hour.values.reshape(-1, 1)
2 X_hour[:10]
```

```
Out[27]: array([[ 0],
   [ 3],
   [ 6],
   [ 9],
   [12],
   [15],
   [18],
   [21],
   [ 0],
   [ 3]])
```

In [28]: 1 eval_on_features(X_hour, y, regressor)

Train-set R²: 0.82
Test-set R²: 0.85

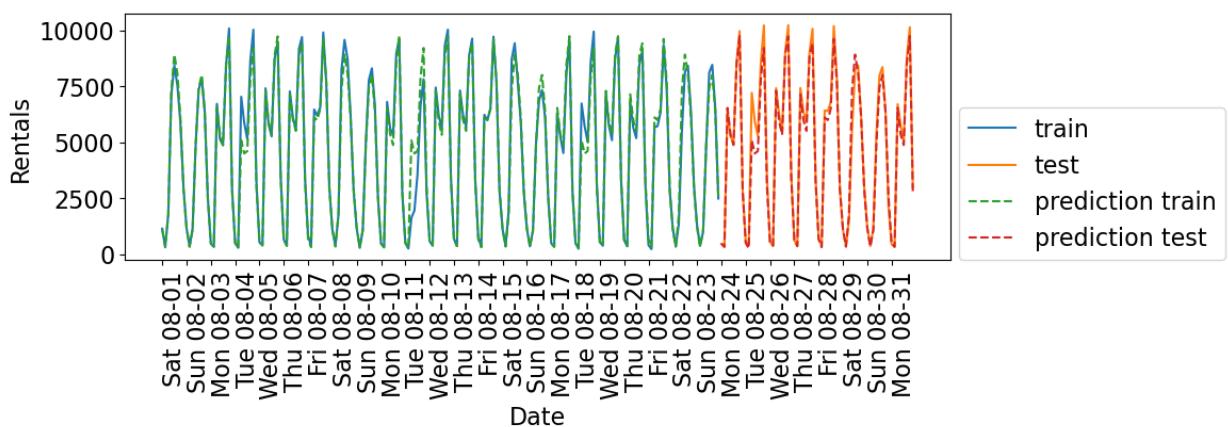


The scores are better than before.

Now let's add day of the week along with time of the day.

In [29]: 1 X_hour_week = np.hstack(
2 [
3 citibike.index.dayofweek.values.reshape(-1, 1),
4 citibike.index.hour.values.reshape(-1, 1),
5]
6)
7 eval_on_features(X_hour_week, y, regressor)

Train-set R²: 0.97
Test-set R²: 0.98



The results are much better. The time of the day and day of the week features are clearly helping.

- Do we need a complex model such as a random forest?
- Let's try Ridge with these features.

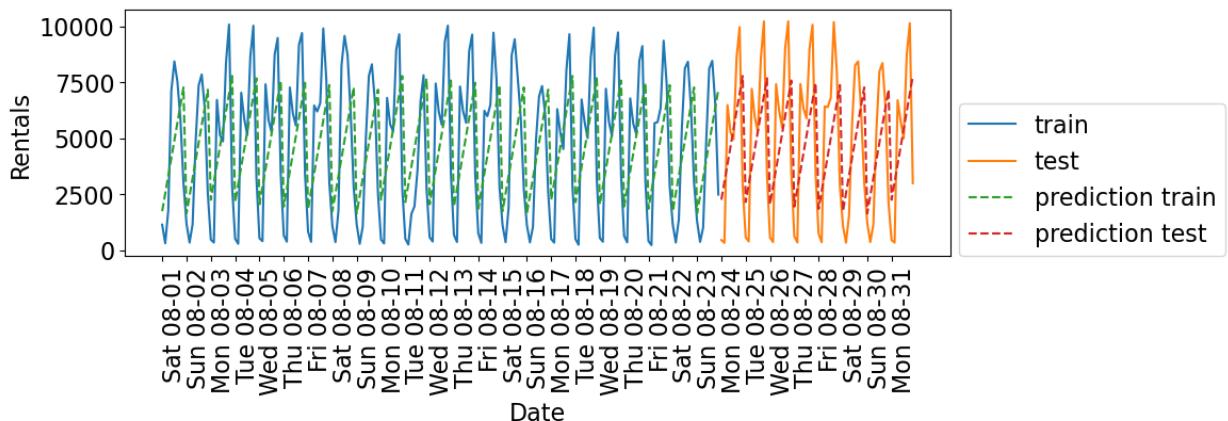
In [30]:

```

1 from sklearn.linear_model import Ridge
2
3 lr = Ridge()
4 eval_on_features(X_hour_week, y, lr)

```

Train-set R²: 0.32
Test-set R²: 0.33



- Ridge is performing poorly on the training as well as test data.
- It's not able to capture the periodic pattern.
- The reason is that we have encoded time of day using integers.
- A linear function can only learn a linear function of the time of day.
- What if we encode this feature as a categorical variable?

In [31]:

```

1 enc = OneHotEncoder()
2 X_hour_week_onehot = enc.fit_transform(X_hour_week).toarray()

```

In [32]:

```

1 X_hour_week_onehot
2 X_hour_week_onehot.shape

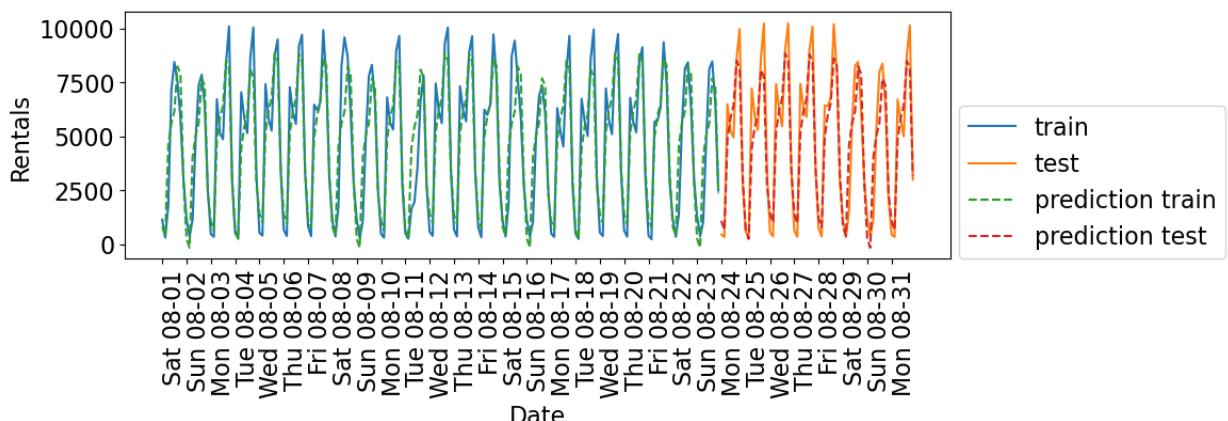
```

Out[32]: (248, 15)

In [33]:

```
1 eval_on_features(X_hour_week_onehot, y, Ridge())
```

Train-set R²: 0.84
Test-set R²: 0.87



What if we add interaction features. We can do it using `sklearn's PolynomialFeatures` transformer.

In [34]:

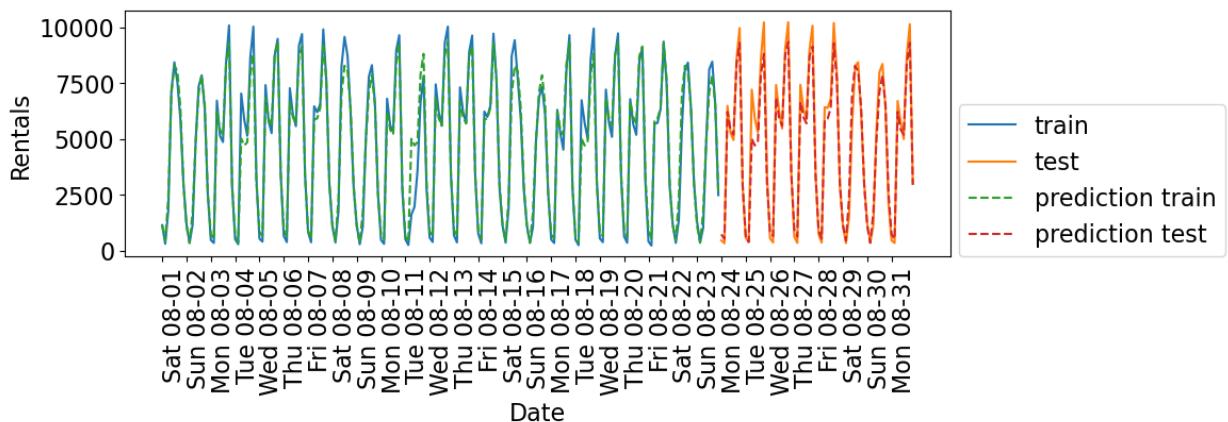
```

1 from sklearn.preprocessing import PolynomialFeatures
2
3 poly_transformer = PolynomialFeatures(
4     degree=2, interaction_only=True, include_bias=False
5 )
6 X_hour_week_onehot_poly = poly_transformer.fit_transform(X_hour_week_on
7 lr = Ridge()
8 eval_on_features(X_hour_week_onehot_poly, y, lr)

```

Train-set R^2: 0.97

Test-set R^2: 0.97



In [35]:

```
1 X_hour_week_onehot_poly.shape
```

Out[35]: (248, 120)

In [36]:

```

1 hour = ["%02d:00" % i for i in range(0, 24, 3)]
2 day = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
3 features = day + hour
4 features

```

Out[36]: ['Mon',
'Tue',
'Wed',
'Thu',
'Fri',
'Sat',
'Sun',
'00:00',
'03:00',
'06:00',
'09:00',
'12:00',
'15:00',
'18:00',
'21:00']

Let's examine the coefficients learned by `Ridge`.

```
In [37]: 1 features_poly = poly_transformer.get_feature_names_out(features)
          2 features_nonzero = np.array(features_poly)[lr.coef_ != 0]
          3 coef_nonzero = lr.coef_[lr.coef_ != 0]
```

```
In [38]: 1 coefs = pd.DataFrame(coef_nonzero, index=features_nonzero, columns=[ "Co
          2      "Coefficient", ascending=False
          3 ])
```

```
In [39]: 1 coefs[:10]
```

Out[39]: **Coefficient**

	Coefficient
15:00	3228.129921
18:00	3012.775591
Sat 12:00	2449.171257
Sun 12:00	1753.500986
Sat 09:00	1568.393304
Wed 06:00	1527.908029
Mon 18:00	1473.445712
Thu 06:00	1380.854458
Tue 18:00	1355.142140
12:00	1251.901575

```
In [40]: 1 coefs[-10:]
```

Out[40]: **Coefficient**

	Coefficient
Thu 00:00	-814.917196
Tue 12:00	-848.702348
Mon 12:00	-884.148776
Wed 00:00	-924.113625
Sat 18:00	-1228.127956
21:00	-1474.846457
Sat 06:00	-2455.263389
Sun 06:00	-2576.733659
00:00	-3353.192913
03:00	-3702.586614

- The coefficients make sense!
- If it's Saturday 09:00 or Wednesday 06:00, the model is likely to predict bigger number for rentals.

In []:

1

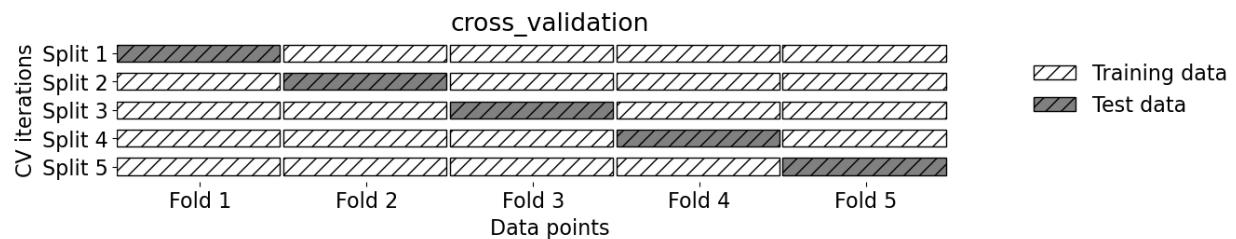
Cross-validation

What about cross-validation?

- We can't do regular cross-validation if we don't want to be predicting the past.
- If you carry out regular cross-validation, you'll be predicting the past given future which is not a realistic scenario for the deployment data.

In [41]:

```
1 import mglearn_utils
2 mglearn_utils.plot_cross_validation()
```



There is [TimeSeriesSplit](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html) (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html) for time series data.

In [42]:

```
1 from sklearn.model_selection import TimeSeriesSplit
```

In [43]:

```
1 # Code from sklearn documentation
2 X_toy = np.array([[1, 2], [3, 4], [1, 2], [3, 4], [1, 2], [3, 4]])
3 y_toy = np.array([1, 2, 3, 4, 5, 6])
4 tscv = TimeSeriesSplit(n_splits=3)
5 for train, test in tscv.split(X_toy):
6     print("%s %s" % (train, test))
```

```
[0 1 2] [3]
[0 1 2 3] [4]
[0 1 2 3 4] [5]
```

Let's try it out with Ridge on the citibike data.

In [44]:

```
1 lr = Ridge()
```

In [45]:

```

1 scores = cross_validate(
2     lr, X_hour_week_onehot_poly, y, cv=TimeSeriesSplit(n_splits=3), ret
3 )
4 pd.DataFrame(scores)

```

Out[45]:

	fit_time	score_time	test_score	train_score
0	0.000553	0.000210	0.807857	0.947222
1	0.000605	0.000139	0.943975	0.946537
2	0.000650	0.000124	0.956777	0.961063
3	0.000667	0.000115	0.953726	0.965518
4	0.000781	0.000115	0.978614	0.967371

A more complicated dataset (5 min)

Rain in Australia (<https://www.kaggle.com/jsphyg/weather-dataset-rattle-package>) dataset.

Predicting whether or not it will rain tomorrow based on today's measurements.

In [46]:

```

1 rain_df = pd.read_csv("../data/weatherAUS.csv")
2 rain_df.head()

```

Out[46]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpe
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	4
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	4
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	4
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	2
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	4

5 rows × 23 columns

In [47]:

```
1 rain_df.shape
```

Out[47]:

(145460, 23)

Goals

- Can the date/time features help us predict the target value?
- Can we **forecast** into the future?

```
In [48]: 1 rain_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Date             145460 non-null   object 
 1   Location         145460 non-null   object 
 2   MinTemp          143975 non-null   float64
 3   MaxTemp          144199 non-null   float64
 4   Rainfall         142199 non-null   float64
 5   Evaporation     82670  non-null    float64
 6   Sunshine         75625 non-null   float64
 7   WindGustDir     135134 non-null   object 
 8   WindGustSpeed   135197 non-null   float64
 9   WindDir9am      134894 non-null   object 
 10  WindDir3pm      141232 non-null   object 
 11  WindSpeed9am    143693 non-null   float64
 12  WindSpeed3pm    142398 non-null   float64
 13  Humidity9am     142806 non-null   float64
 14  Humidity3pm     140953 non-null   float64
 15  Pressure9am     130395 non-null   float64
 16  Pressure3pm     130432 non-null   float64
 17  Cloud9am         89572  non-null    float64
 18  Cloud3pm         86102  non-null    float64
 19  Temp9am          143693 non-null   float64
 20  Temp3pm          141851 non-null   float64
 21  RainToday        142199 non-null   object 
 22  RainTomorrow     142193 non-null   object 

dtypes: float64(16), object(7)
memory usage: 25.5+ MB
```

In [49]: 1 rain_df.describe(include="all")

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
count	145460	145460	143975.000000	144199.000000	142199.000000	82670.000000	75625.000000
unique	3436	49	NaN	NaN	NaN	NaN	NaN
top	2013-11-12	Canberra	NaN	NaN	NaN	NaN	NaN
freq	49	3436	NaN	NaN	NaN	NaN	NaN
mean	NaN	NaN	12.194034	23.221348	2.360918	5.468232	7.611178
std	NaN	NaN	6.398495	7.119049	8.478060	4.193704	3.785483
min	NaN	NaN	-8.500000	-4.800000	0.000000	0.000000	0.000000
25%	NaN	NaN	7.600000	17.900000	0.000000	2.600000	4.800000
50%	NaN	NaN	12.000000	22.600000	0.000000	4.800000	8.400000
75%	NaN	NaN	16.900000	28.200000	0.800000	7.400000	10.600000
max	NaN	NaN	33.900000	48.100000	371.000000	145.000000	14.500000

11 rows × 23 columns

- A number of missing values.
- Some target values are also missing. I'm dropping these rows.

In [50]: 1 rain_df = rain_df[rain_df["RainTomorrow"].notna()]
2 rain_df.shape

Out[50]: (142193, 23)

Parsing datetimes

- In general, datetimes are a huge pain! Think of all the formats: MM-DD-YY, DD-MM-YY, YY-MM-DD, MM/DD/YY, DD/MM/YY, DD/MM/YYYY, 20:45, 8:45am, 8:45 PM, 8:45a, 08:00, 8:10:20,
- No, seriously, dealing with datetimes is THE WORST.
 - Time zones.
 - Daylight savings...
- Thankfully, pandas does a pretty good job here.

```
In [51]: 1 dates_rain = pd.to_datetime(rain_df["Date"])
          2 dates_rain
```

```
Out[51]: 0      2008-12-01
          1      2008-12-02
          2      2008-12-03
          3      2008-12-04
          4      2008-12-05
          ...
          145454  2017-06-20
          145455  2017-06-21
          145456  2017-06-22
          145457  2017-06-23
          145458  2017-06-24
Name: Date, Length: 142193, dtype: datetime64[ns]
```

They are all the same format, so we can also compare dates:

```
In [52]: 1 dates_rain[1] - dates_rain[0]
```

```
Out[52]: Timedelta('1 days 00:00:00')
```

```
In [53]: 1 dates_rain[1] > dates_rain[0]
```

```
Out[53]: True
```

```
In [54]: 1 (dates_rain[1] - dates_rain[0]).total_seconds()
```

```
Out[54]: 86400.0
```

We can also easily extract information from the date columns.

```
In [55]: 1 dates_rain[1]
```

```
Out[55]: Timestamp('2008-12-02 00:00:00')
```

```
In [56]: 1 dates_rain[1].month_name()
```

```
Out[56]: 'December'
```

```
In [57]: 1 dates_rain[1].day_name()
```

```
Out[57]: 'Tuesday'
```

```
In [58]: 1 dates_rain[1].is_year_end
```

```
Out[58]: False
```

```
In [59]: 1 dates_rain[1].is_leap_year
```

```
Out[59]: True
```

Above pandas identified the date column automatically. You can tell pandas to parse the dates when reading in the CSV:

```
In [60]: 1 rain_df = pd.read_csv("../data/weatherAUS.csv", parse_dates=[ "Date" ])
2 rain_df.head()
```

```
Out[60]:
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpe
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	4
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	4
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	4
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	2
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	4

5 rows × 23 columns

```
In [61]: 1 rain_df = rain_df[rain_df[ "RainTomorrow" ].notna()]
2 rain_df.shape
```

```
Out[61]: (142193, 23)
```

```
In [62]: 1 rain_df[ "Date" ].head()
```

```
Out[62]: 0    2008-12-01
1    2008-12-02
2    2008-12-03
3    2008-12-04
4    2008-12-05
Name: Date, dtype: datetime64[ns]
```

Train/test splits

- Remember that we should not be calling the usual `train_test_split` with shuffling because
- If we want to forecast, we aren't allowed to know what happened in the future!

```
In [63]: 1 rain_df[ "Date" ].min()
```

```
Out[63]: Timestamp('2007-11-01 00:00:00')
```

```
In [64]: 1 rain_df[ "Date" ].max()
```

```
Out[64]: Timestamp('2017-06-25 00:00:00')
```

- It looks like we have 10 years of data.

- Let's use the last 2 years for test.

```
In [65]: 1 train_df = rain_df.query("Date <= 20150630")
          2 test_df = rain_df.query("Date > 20150630")
```

```
In [66]: 1 len(train_df)
```

```
Out[66]: 107502
```

```
In [67]: 1 len(test_df)
```

```
Out[67]: 34691
```

```
In [68]: 1 len(test_df) / (len(train_df) + len(test_df))
```

```
Out[68]: 0.24397122221206388
```

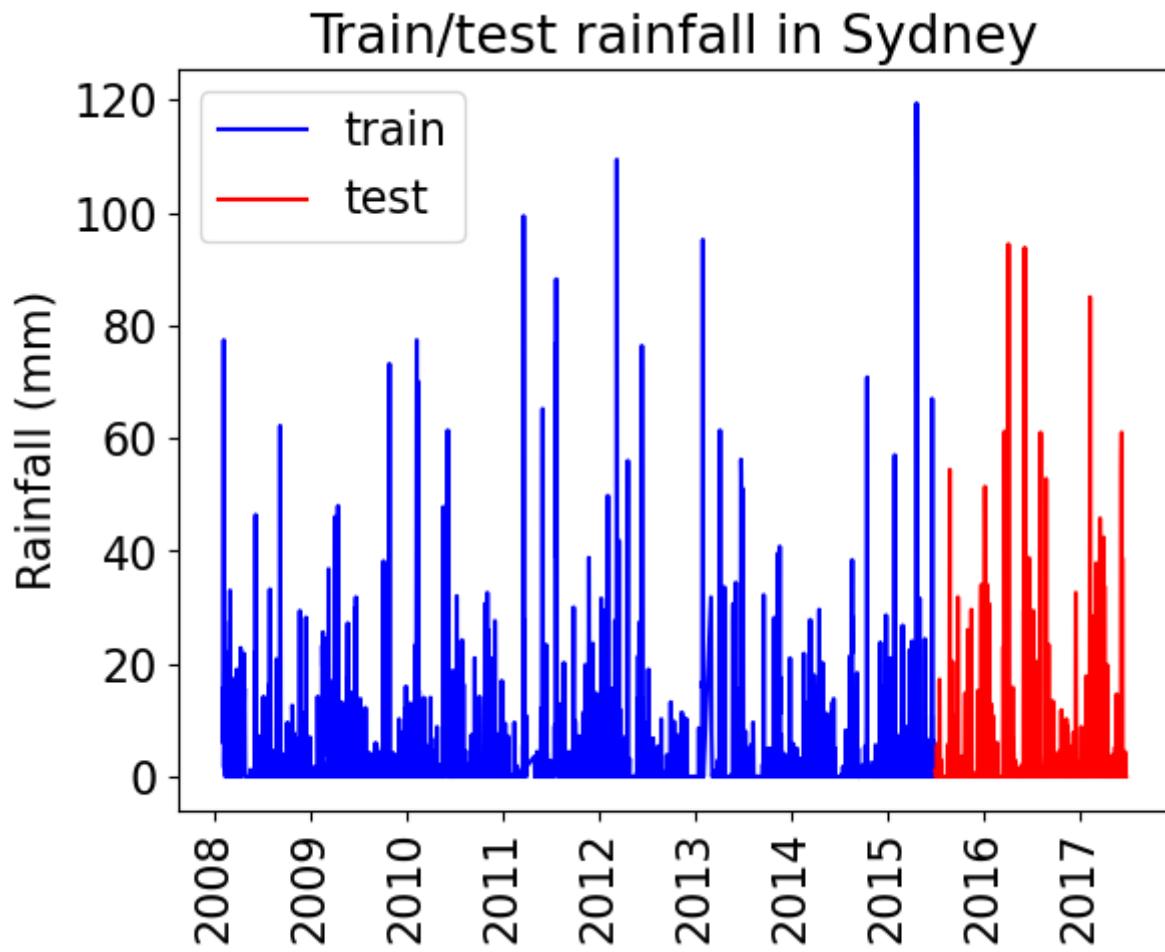
As we can see, we're still using about 25% of our data as test data.

In [69]:

```

1 train_df_sort = train_df.query("Location == 'Sydney'").sort_values(by="Date")
2 test_df_sort = test_df.query("Location == 'Sydney'").sort_values(by="Date")
3
4 plt.plot(train_df_sort["Date"], train_df_sort["Rainfall"], "b", label="train")
5 plt.plot(test_df_sort["Date"], test_df_sort["Rainfall"], "r", label="test")
6 plt.xticks(rotation="vertical")
7 plt.legend()
8 plt.ylabel("Rainfall (mm)")
9 plt.title("Train/test rainfall in Sydney");

```



We're learning relationships from the blue part; predicting only using features in the red part from the day before.

Let's define a preprocessor with a column transformer.

In [70]:

```
1 train_df.columns
```

Out[70]:

```
Index(['Date', 'Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation',
       'Sunshine', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am',
       'Temp3pm', 'RainToday', 'RainTomorrow'],
      dtype='object')
```

In [71]: 1 train_df.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 107502 entries, 0 to 144733
Data columns (total 23 columns):
 #   Column           Non-Null Count   Dtype  
--- 
 0   Date              107502 non-null    datetime64[ns] 
 1   Location          107502 non-null    object  
 2   MinTemp           107050 non-null    float64 
 3   MaxTemp           107292 non-null    float64 
 4   Rainfall          106424 non-null    float64 
 5   Evaporation       66221 non-null     float64 
 6   Sunshine          62320 non-null     float64 
 7   WindGustDir       100103 non-null    object  
 8   WindGustSpeed    100146 non-null    float64 
 9   WindDir9am        99515 non-null     object  
 10  WindDir3pm        105314 non-null    object  
 11  WindSpeed9am     106322 non-null    float64 
 12  WindSpeed3pm     106319 non-null    float64 
 13  Humidity9am      106112 non-null    float64 
 14  Humidity3pm      106180 non-null    float64 
 15  Pressure9am      97217 non-null     float64 
 16  Pressure3pm      97253 non-null     float64 
 17  Cloud9am          68523 non-null     float64 
 18  Cloud3pm          67501 non-null     float64 
 19  Temp9am           106705 non-null    float64 
 20  Temp3pm           106816 non-null    float64 
 21  RainToday          106424 non-null    object  
 22  RainTomorrow       107502 non-null    object  
dtypes: datetime64[ns](1), float64(16), object(6)
memory usage: 19.7+ MB
```

- We have missing data.
- We have categorical features and numeric features.

- Let's define feature types.
- Let's start with dropping the date column and treating it as a usual supervised machine learning problem.

In [72]:

```
1 numeric_features = [
2     "MinTemp",
3     "MaxTemp",
4     "Rainfall",
5     "Evaporation",
6     "Sunshine",
7     "WindGustSpeed",
8     "WindSpeed9am",
9     "WindSpeed3pm",
10    "Humidity9am",
11    "Humidity3pm",
12    "Pressure9am",
13    "Pressure3pm",
14    "Cloud9am",
15    "Cloud3pm",
16    "Temp9am",
17    "Temp3pm",
18 ]
19 categorical_features = [
20     "Location",
21     "WindGustDir",
22     "WindDir9am",
23     "WindDir3pm",
24     "RainToday",
25 ]
26 drop_features = [
27     "Date",
28     "RainTomorrow",
29 ]
```

In [73]:

```
1 def preprocess_features(
2     train_df,
3     test_df,
4     numeric_features,
5     categorical_features,
6     drop_features,
7 ):
8
9     all_features = set(numeric_features + categorical_features + drop_f
10    if set(train_df.columns) != all_features:
11        print("Missing columns", set(train_df.columns) - all_features)
12        print("Extra columns", all_features - set(train_df.columns))
13        raise Exception("Columns do not match")
14
15    numeric_transformer = make_pipeline(
16        SimpleImputer(strategy="median"), StandardScaler()
17    )
18    categorical_transformer = make_pipeline(
19        SimpleImputer(strategy="constant", fill_value="?"),
20        OneHotEncoder(handle_unknown="ignore", sparse=False),
21    )
22
23    preprocessor = make_column_transformer(
24        (numeric_transformer, numeric_features),
25        (categorical_transformer, categorical_features),
26        ("drop", drop_features),
27    )
28    preprocessor.fit(train_df)
29    ohe_feature_names = (
30        preprocessor.named_transformers_["pipeline-2"]
31        .named_steps["onehotencoder"]
32        .get_feature_names_out()
33        .tolist()
34    )
35    new_columns = numeric_features + ohe_feature_names
36
37    X_train_enc = pd.DataFrame(
38        preprocessor.transform(train_df), index=train_df.index, columns=
39    )
40    X_test_enc = pd.DataFrame(
41        preprocessor.transform(test_df), index=test_df.index, columns=n
42    )
43
44    y_train = train_df["RainTomorrow"]
45    y_test = test_df["RainTomorrow"]
46
47    return X_train_enc, y_train, X_test_enc, y_test, preprocessor
```

```
In [74]: 1 X_train_enc, y_train, X_test_enc, y_test, preprocessor = preprocess_fea
          2     train_df,
          3     test_df,
          4     numeric_features,
          5     categorical_features,
          6     drop_features,
          7 )
```

```
In [75]: 1 X_train_enc.head()
```

Out[75]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSp
0	0.204302	-0.027112	-0.205323	-0.140641	0.160729	0.298612	0.666166	0
1	-0.741037	0.287031	-0.275008	-0.140641	0.160729	0.298612	-1.125617	0
2	0.125523	0.372706	-0.275008	-0.140641	0.160729	0.450132	0.554180	0
3	-0.457435	0.701128	-0.275008	-0.140641	0.160729	-1.216596	-0.341712	-1
4	0.850283	1.315134	-0.158867	-0.140641	0.160729	0.071330	-0.789657	0

5 rows × 119 columns

Baseline

DummyClassifier

```
In [76]: 1 dc = DummyClassifier(strategy="prior")
          2 dc.fit(train_df, y_train);
```

```
In [77]: 1 dc.score(train_df, y_train)
```

Out[77]: 0.7750553478074826

```
In [78]: 1 y_train.value_counts()
```

Out[78]: No 83320
Yes 24182
Name: RainTomorrow, dtype: int64

```
In [79]: 1 dc.score(test_df, y_test)
```

Out[79]: 0.7781845435415525

LogisticRegression

The function below trains a logistic regression model on the train set, reports train and test scores, and returns learned coefficients as a dataframe.

In [80]:

```

1 def score_lr_print_coeff(preprocessor, train_df, y_train, test_df, y_te
2     lr_pipe = make_pipeline(preprocessor, LogisticRegression(max_iter=1
3     lr_pipe.fit(train_df, y_train)
4     print("Train score: {:.2f}".format(lr_pipe.score(train_df, y_train)))
5     print("Test score: {:.2f}".format(lr_pipe.score(test_df, y_test)))
6     lr_coef = pd.DataFrame(
7         data=lr_pipe.named_steps["logisticregression"].coef_.flatten(),
8         index=X_train_enc.columns,
9         columns=["Coef"],
10    )
11    return lr_coef.sort_values(by="Coef", ascending=False)

```

In [81]:

```
1 score_lr_print_coeff(preprocessor, train_df, y_train, test_df, y_test,
```

Train score: 0.85
 Test score: 0.84

Out[81]:

	Coef
Humidity3pm	1.243231
x4_?	0.924657
Pressure9am	0.865428
x0_Witchcliffe	0.729015
WindGustSpeed	0.720465
...	...
x0_Townsville	-0.718734
x0_Katherine	-0.726151
x0_Wollongong	-0.748688
x0_MountGinini	-0.964870
Pressure3pm	-1.221746

119 rows × 1 columns

Cross-validation

- We can carry out cross-validation using [TimeSeriesSplit \(\[https://scikit-learn.org/stable/modules/cross_validation.html#time-series-split\]\(https://scikit-learn.org/stable/modules/cross_validation.html#time-series-split\)\)](https://scikit-learn.org/stable/modules/cross_validation.html#time-series-split).
- However, things are actually more complicated here because this dataset has **multiple time series**, one per location.

In [82]: 1 train_df

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGu
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	
...
144729	2015-06-26	Uluru	3.8	18.3	0.0	NaN	NaN	E	
144730	2015-06-27	Uluru	2.5	17.1	0.0	NaN	NaN	E	
144731	2015-06-28	Uluru	4.5	19.6	0.0	NaN	NaN	ENE	
144732	2015-06-29	Uluru	7.6	22.0	0.0	NaN	NaN	ESE	
144733	2015-06-30	Uluru	6.8	21.1	0.0	NaN	NaN	ESE	

107502 rows × 23 columns

In [83]: 1 train_df.sort_values(by=["Date", "Location"]).head()

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGus
45587	2007-11-01	Canberra	8.0	24.3	0.0	3.4	6.3	NW	
45588	2007-11-02	Canberra	14.0	26.9	3.6	4.4	9.7	ENE	
45589	2007-11-03	Canberra	13.7	23.4	3.6	5.8	3.3	NW	
45590	2007-11-04	Canberra	13.3	15.5	39.8	7.2	9.1	NW	
45591	2007-11-05	Canberra	7.6	16.1	2.8	5.6	10.6	SSE	

5 rows × 23 columns

In [84]: 1 train_df.sort_values(by= "Date")

Out[84]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir
45587	2007-11-01	Canberra	8.0	24.3	0.0	3.4	6.3	NW
45588	2007-11-02	Canberra	14.0	26.9	3.6	4.4	9.7	ENE
45589	2007-11-03	Canberra	13.7	23.4	3.6	5.8	3.3	NW
45590	2007-11-04	Canberra	13.3	15.5	39.8	7.2	9.1	NW
45591	2007-11-05	Canberra	7.6	16.1	2.8	5.6	10.6	SSE
...
57415	2015-06-30	Ballarat	-0.3	10.5	0.0	NaN	NaN	S
119911	2015-06-30	PerthAirport	10.1	23.5	0.0	3.2	5.8	NNE
60455	2015-06-30	Bendigo	0.3	11.4	0.0	NaN	NaN	W
66473	2015-06-30	MelbourneAirport	3.2	13.2	0.0	0.8	3.9	N
144733	2015-06-30	Uluru	6.8	21.1	0.0	NaN	NaN	ESE

107502 rows × 23 columns

- It seems the dataframe is sorted by location, and then time.
- Our approach today will be to ignore the fact that we have multiple time series and just OHE the location
- We'll have multiple measurements for a given timestamp, and that's OK.
- But, `TimeSeriesSplit` expects the dataframe to be sorted by date so...

In [85]: 1 train_df_ordered = train_df.sort_values(by= "Date")
2 y_train_ordered = train_df_ordered["RainTomorrow"]

In [86]: 1 train_df_ordered

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir
45587	2007-11-01	Canberra	8.0	24.3	0.0	3.4	6.3	NW
45588	2007-11-02	Canberra	14.0	26.9	3.6	4.4	9.7	ENE
45589	2007-11-03	Canberra	13.7	23.4	3.6	5.8	3.3	NW
45590	2007-11-04	Canberra	13.3	15.5	39.8	7.2	9.1	NW
45591	2007-11-05	Canberra	7.6	16.1	2.8	5.6	10.6	SSE
...
57415	2015-06-30	Ballarat	-0.3	10.5	0.0	NaN	NaN	S
119911	2015-06-30	PerthAirport	10.1	23.5	0.0	3.2	5.8	NNE
60455	2015-06-30	Bendigo	0.3	11.4	0.0	NaN	NaN	W
66473	2015-06-30	MelbourneAirport	3.2	13.2	0.0	0.8	3.9	N
144733	2015-06-30	Uluru	6.8	21.1	0.0	NaN	NaN	ESE

107502 rows × 23 columns

In [87]: 1 lr_pipe = make_pipeline(preprocessor, LogisticRegression(max_iter=1000)
2 cross_val_score(lr_pipe, train_df_ordered, y_train_ordered, cv=TimeSeri

Out[87]: 0.8478874811631412

In []:

Encoding date/time as feature(s)

- Can we use the Date to help us predict the target?
- Probably! E.g. different amounts of rain in different seasons.
- This is feature engineering!

Encoding time as an number

- Idea 1: create a column of "days since Nov 1, 2007".

```
In [88]: 1 train_df = rain_df.query("Date <= 20150630")
2 test_df = rain_df.query("Date > 20150630")
```

```
In [89]: 1 first_day = train_df["Date"].min()
2
3 train_df = train_df.assign(
4     Days_since=train_df["Date"].apply(lambda x: (x - first_day).days)
5 )
6 test_df = test_df.assign(
7     Days_since=test_df["Date"].apply(lambda x: (x - first_day).days)
8 )
```

```
In [90]: 1 train_df.sort_values(by="Date").head()
```

Out[90]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGus
45587	2007-11-01	Canberra	8.0	24.3	0.0	3.4	6.3	NW	
45588	2007-11-02	Canberra	14.0	26.9	3.6	4.4	9.7	ENE	
45589	2007-11-03	Canberra	13.7	23.4	3.6	5.8	3.3	NW	
45590	2007-11-04	Canberra	13.3	15.5	39.8	7.2	9.1	NW	
45591	2007-11-05	Canberra	7.6	16.1	2.8	5.6	10.6	SSE	

5 rows × 24 columns

```
In [91]: 1 X_train_enc, y_train, X_test_enc, y_test, preprocessor = preprocess_fea
2     train_df,
3     test_df,
4     numeric_features + ["Days_since"],
5     categorical_features,
6     drop_features,
7 )
```

```
In [92]: 1 score_lr_print_coeff(preprocessor, train_df, y_train, test_df, y_test,
```

Train score: 0.85
Test score: 0.84

Out[92]:

	Coef
Humidity3pm	1.243081
x4_?	0.931186
Pressure9am	0.864258
x0_Witchcliffe	0.730547
WindGustSpeed	0.720084
...	...
x0_Townsville	-0.716479
x0_Katherine	-0.737178
x0_Wollongong	-0.746131
x0_MountGinini	-0.963908
Pressure3pm	-1.221575

120 rows × 1 columns

- Not much improvement in the scores
- Can you think of other ways to generate features from the `Date` column?
- What about the month - that seems relevant. How should we encode the month?

Another idea month: encode as a categorical variable?

One-hot encoding of the month

```
In [93]: 1 train_df = rain_df.query("Date <= 20150630")
2 test_df = rain_df.query("Date > 20150630")
```

```
In [94]: 1 train_df = train_df.assign(
2     Month=train_df["Date"].apply(lambda x: x.month_name())
3 ) # x.month_name() to get the actual string
4 test_df = test_df.assign(Month=test_df["Date"].apply(lambda x: x.month_
```

```
In [95]: 1 train_df[["Date", "Month"]].sort_values(by="Month")
```

Out[95]:

	Date	Month
62657	2013-04-14	April
115089	2010-04-15	April
115090	2010-04-16	April
115091	2010-04-17	April
115092	2010-04-18	April
...
128828	2014-09-20	September
128829	2014-09-21	September
128830	2014-09-22	September
128820	2014-09-12	September
72036	2013-09-29	September

107502 rows × 2 columns

```
In [96]: 1 X_train_enc, y_train, X_test_enc, y_test, preprocessor = preprocess_fea
2     train_df, test_df, numeric_features, categorical_features + ["Month"]
3 )
```

```
In [97]: 1 score_lr_print_coeff(preprocessor, train_df, y_train, test_df, y_test,
```

Train score: 0.85
Test score: 0.84

Out[97]:

	Coef
Humidity3pm	1.266933
x4_?	0.944802
Pressure9am	0.799138
x0_Witchcliffe	0.749015
WindGustSpeed	0.705677
...	...
x0_Darwin	-0.735810
x0_Wollongong	-0.748377
x0_Townsville	-0.903095
x0_Katherine	-0.929008
Pressure3pm	-1.182011

131 rows × 1 columns

One-hot encoding seasons

How about just summer/winter as a feature?

In [98]:

```

1 def get_season(month):
2     WINTER_MONTHS = ["June", "July", "August"]
3     AUTUMN_MONTHS = ["March", "April", "May"]
4     SUMMER_MONTHS = ["December", "January", "February"]
5     SPRING_MONTHS = ["September", "October", "November"]
6     if month in WINTER_MONTHS:
7         return "Winter"
8     elif month in AUTUMN_MONTHS:
9         return "Autumn"
10    elif month in SUMMER_MONTHS:
11        return "Summer"
12    else:
13        return "Fall"

```

In [99]:

```

1 train_df = train_df.assign(Season=train_df["Month"].apply(get_season))
2 test_df = test_df.assign(Season=test_df["Month"].apply(get_season))

```

In [100]:

```
1 train_df
```

Out[100]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGu
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	
...
144729	2015-06-26	Uluru	3.8	18.3	0.0	NaN	NaN	E	
144730	2015-06-27	Uluru	2.5	17.1	0.0	NaN	NaN	E	
144731	2015-06-28	Uluru	4.5	19.6	0.0	NaN	NaN	ENE	
144732	2015-06-29	Uluru	7.6	22.0	0.0	NaN	NaN	ESE	
144733	2015-06-30	Uluru	6.8	21.1	0.0	NaN	NaN	ESE	

107502 rows × 25 columns

```
In [101]: 1 X_train_enc, y_train, X_test_enc, y_test, preprocessor = preprocess_fea
          2     train_df,
          3     test_df,
          4     numeric_features,
          5     categorical_features + ["Season"],
          6     drop_features + ["Month"],
          7 )
```

```
In [102]: 1 X_train_enc.columns
```

```
Out[102]: Index(['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine',
       'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am',
       'Humidity3pm',
       ...
       'x3_W', 'x3_WNW', 'x3_WSW', 'x4_?', 'x4_No', 'x4_Yes', 'x5_Autum
n',
       'x5_Fall', 'x5_Summer', 'x5_Winter'],
      dtype='object', length=123)
```

```
In [103]: 1 coeff_df = score_lr_print_coeff(
          2     preprocessor, train_df, y_train, test_df, y_test, X_train_enc
          3 )
```

Train score: 0.85
Test score: 0.84

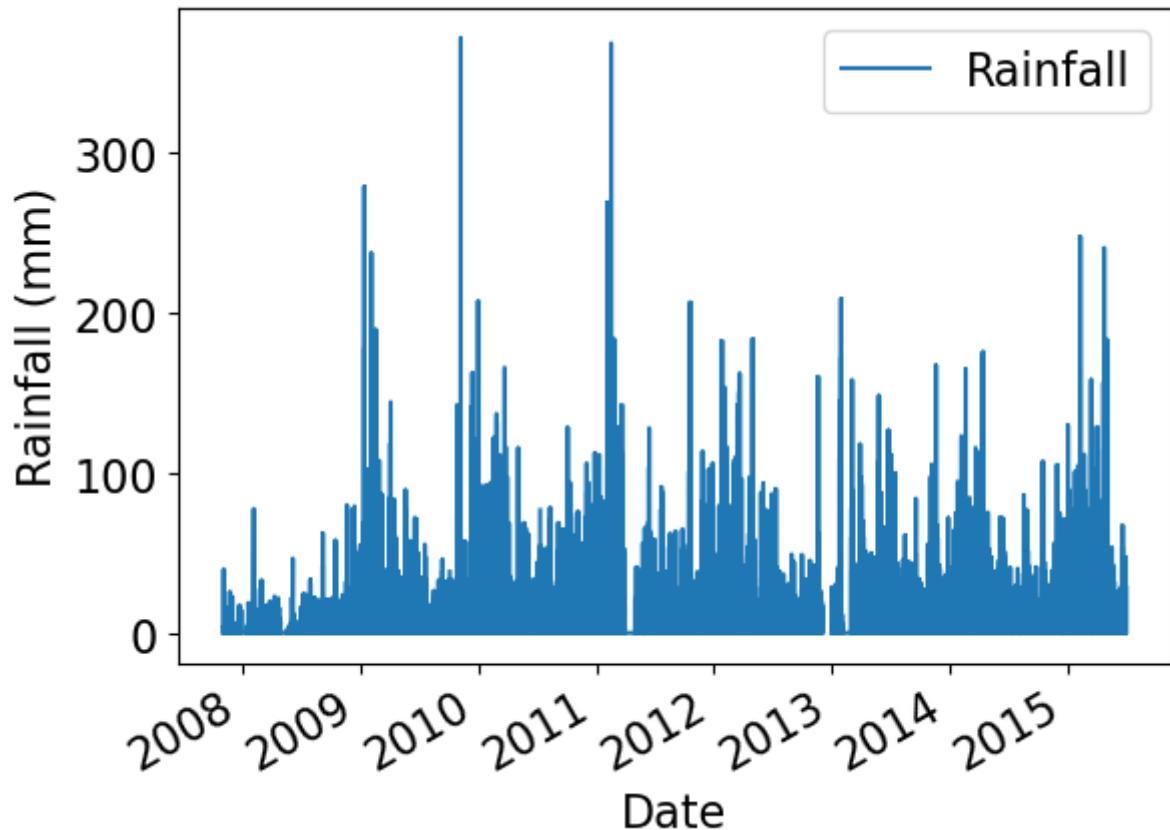
```
In [104]: 1 coeff_df.loc[["x5_Fall", "x5_Summer", "x5_Winter", "x5_Autumn"]]
```

```
Out[104]: Coef
_____
x5_Fall  0.068050
x5_Summer -0.221049
x5_Winter  0.109448
x5_Autumn  0.049084
```

- No improvements in the scores but the coefficients make some sense,
- A negative coefficient for summer and a positive coefficients for winter.

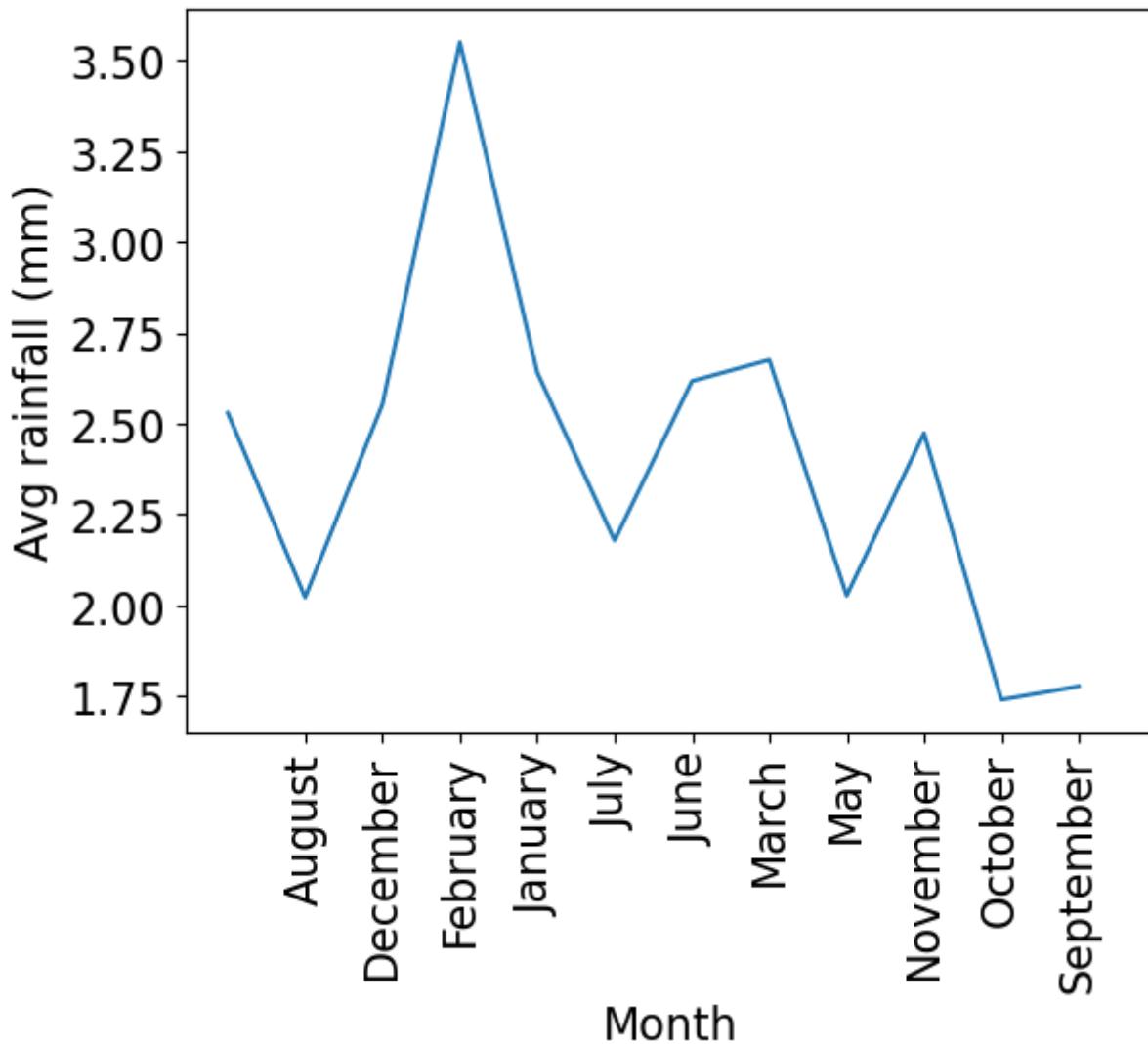
In [105]:

```
1 train_df.plot(x="Date", y="Rainfall")
2 plt.ylabel("Rainfall (mm)");
```



In [106]:

```
1 monthly_avg_rainfall = train_df.groupby("Month")["Rainfall"].mean()
2 plt.plot(monthly_avg_rainfall)
3 plt.xticks(np.arange(1, 13).astype(int))
4 plt.ylabel("Avg rainfall (mm)")
5 plt.xlabel("Month")
6 plt.xticks(rotation=90);
```



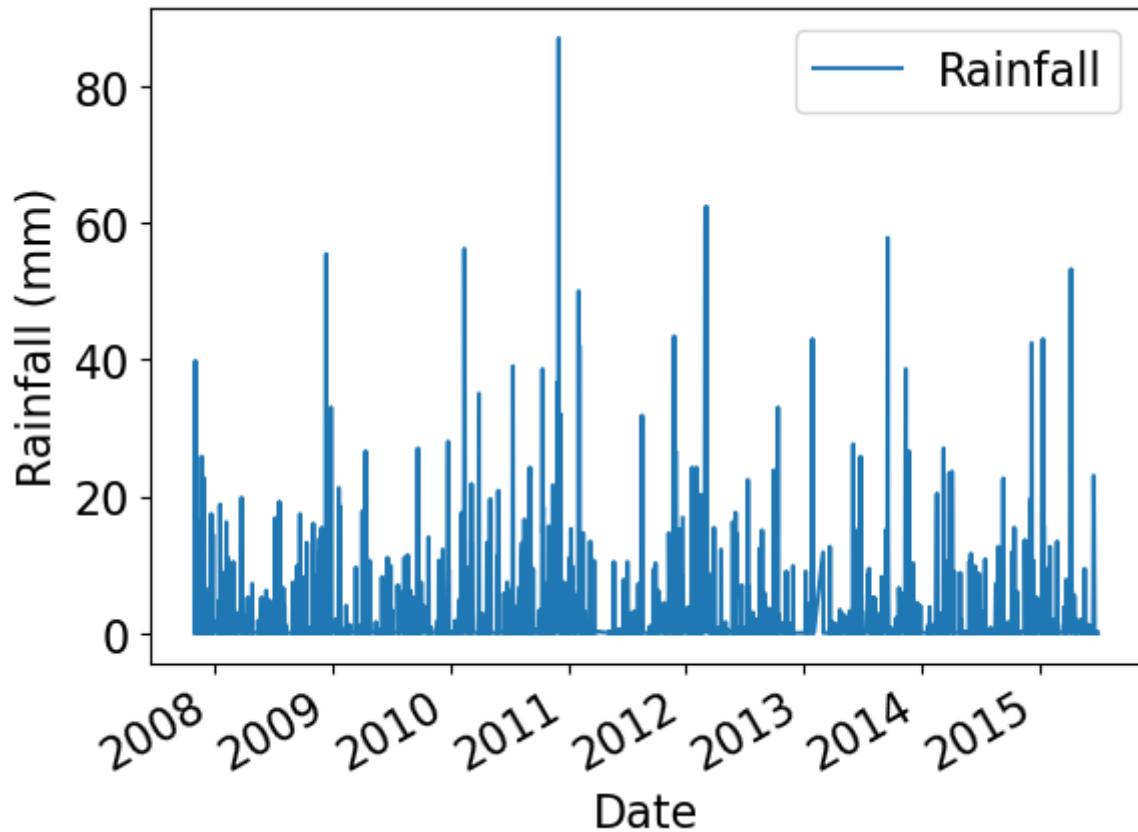
- It's interesting that June rainy but May and August are less so.
- But, Australia is a huge country. Perhaps we should drill down to particular locations:

In [107]:

```
1 train_df_canberra = train_df.query('Location == "Canberra"')
```

In [108]:

```
1 train_df_canberra.plot(x="Date", y="Rainfall")
2 plt.ylabel("Rainfall (mm)");
```

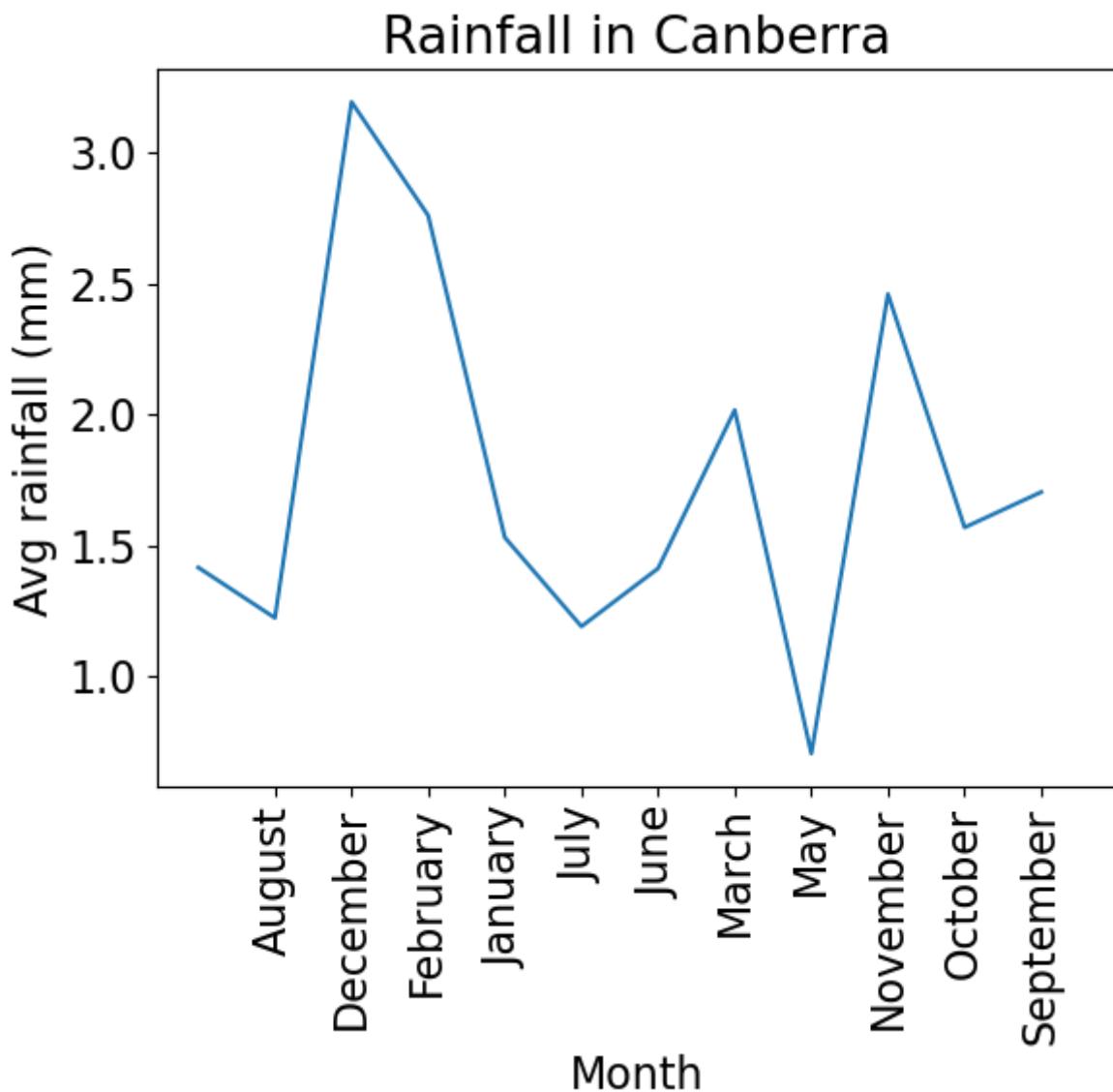


In [109]:

```

1 plt.plot(train_df_canberra.groupby("Month")["Rainfall"].mean())
2 plt.xticks(np.arange(1, 13).astype(int))
3 plt.ylabel("Avg rainfall (mm)")
4 plt.xlabel("Month")
5 plt.title("Rainfall in Canberra")
6 plt.xticks(rotation=90);

```



In [110]:

```
1 train_df_canberra.shape
```

Out[110]: (2696, 25)

- This looks somewhat cleaner but also pretty surprising - why is December so much higher than January?

Lag-based features

- In time series data there is temporal dependence; observations close in time tend to be correlated.

- Currently we're using features about today to predict tomorrow's rainfall.
- But, what if tomorrow's rainfall is also related to yesterday's features, or the day before?
 - This is called a *lagged* feature.
- In time series analysis, we'd look at something called an [autocorrelation function](https://en.wikipedia.org/wiki/Autocorrelation) (<https://en.wikipedia.org/wiki/Autocorrelation>) (ACF), but we won't go into that here.
- Instead, we can just add those features:

```
In [111]: 1 train_df = rain_df.query("Date <= 20150630")
2 test_df = rain_df.query("Date > 20150630")
```

```
In [112]: 1 train_df
```

Out[112]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGu
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	
...	
144729	2015-06-26	Uluru	3.8	18.3	0.0	NaN	NaN	E	
144730	2015-06-27	Uluru	2.5	17.1	0.0	NaN	NaN	E	
144731	2015-06-28	Uluru	4.5	19.6	0.0	NaN	NaN	ENE	
144732	2015-06-29	Uluru	7.6	22.0	0.0	NaN	NaN	ESE	
144733	2015-06-30	Uluru	6.8	21.1	0.0	NaN	NaN	ESE	

107502 rows × 23 columns

- It looks like the dataframe is already sorted by Location and then by date for each Location.
- We could have done this ourselves with:

```
In [113]: 1 # train_df.sort_values(by=[ "Location", "Date"])
```

But make sure to also sort the targets (i.e. do this before preprocessing).

We can "lag" (or "shift") a time series in Pandas with the .shift() method.

```
In [114]: 1 train_df = train_df.assign(Rainfall_lag1=train_df["Rainfall"].shift(1))
```

```
In [115]: 1 train_df[["Date", "Location", "Rainfall", "Rainfall_lag1"]][:20]
```

Out[115]:

	Date	Location	Rainfall	Rainfall_lag1
0	2008-12-01	Albury	0.6	NaN
1	2008-12-02	Albury	0.0	0.6
2	2008-12-03	Albury	0.0	0.0
3	2008-12-04	Albury	0.0	0.0
4	2008-12-05	Albury	1.0	0.0
5	2008-12-06	Albury	0.2	1.0
6	2008-12-07	Albury	0.0	0.2
7	2008-12-08	Albury	0.0	0.0
8	2008-12-09	Albury	0.0	0.0
9	2008-12-10	Albury	1.4	0.0
10	2008-12-11	Albury	0.0	1.4
11	2008-12-12	Albury	2.2	0.0
12	2008-12-13	Albury	15.6	2.2
13	2008-12-14	Albury	3.6	15.6
15	2008-12-16	Albury	NaN	3.6
16	2008-12-17	Albury	0.0	NaN
17	2008-12-18	Albury	16.8	0.0
18	2008-12-19	Albury	10.6	16.8
19	2008-12-20	Albury	0.0	10.6
20	2008-12-21	Albury	0.0	0.0

- But we have multiple time series here and we need to be more careful with this.
- When we switch from one location to another we do not want to take the value from the previous location.

In [116]:

```

1 def create_lag_feature(df, orig_feature, lag):
2     """Creates a new df with a new feature that's a lagged version of t
3     # note: pandas .shift() kind of does this for you already, but oh w
4
5     new_df = df.copy()
6     new_feature_name = "%s_lag%d" % (orig_feature, lag)
7     new_df[new_feature_name] = np.nan
8     for location, df_location in new_df.groupby(
9         "Location"
10    ): # Each location is its own time series
11        new_df.loc[df_location.index[lag:], new_feature_name] = df_loca
12            orig_feature
13        ].values
14    return new_df

```

In [117]:

```
1 train_df = create_lag_feature(train_df, "Rainfall", 1)
```

In [118]:

```
1 train_df[["Date", "Location", "Rainfall", "Rainfall_lag1"]][2285:2295]
```

Out[118]:

	Date	Location	Rainfall	Rainfall_lag1
2309	2015-06-26	Albury	0.2	1.0
2310	2015-06-27	Albury	0.0	0.2
2311	2015-06-28	Albury	0.2	0.0
2312	2015-06-29	Albury	0.0	0.2
2313	2015-06-30	Albury	0.0	0.0
3040	2009-01-01	BadgerysCreek	0.0	NaN
3041	2009-01-02	BadgerysCreek	0.0	0.0
3042	2009-01-03	BadgerysCreek	0.0	0.0
3043	2009-01-04	BadgerysCreek	0.0	0.0
3044	2009-01-05	BadgerysCreek	0.0	0.0

Now it looks good!

Question: is it OK to do this to the test set? Discuss.

- It's fine if you would have this information available in deployment.
- If we're just forecasting the next day, we should.
- Let's include it for now.

In [119]:

```

1 rain_df_modified = create_lag_feature(rain_df, "Rainfall", 1)
2 train_df = rain_df_modified.query("Date <= 20150630")
3 test_df = rain_df_modified.query("Date > 20150630")

```

In [120]: 1 rain_df_modified

Out[120]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGu
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	
...
145454	2017-06-20	Uluru	3.5	21.8	0.0	NaN	NaN	E	
145455	2017-06-21	Uluru	2.8	23.4	0.0	NaN	NaN	E	
145456	2017-06-22	Uluru	3.6	25.3	0.0	NaN	NaN	NNW	
145457	2017-06-23	Uluru	5.4	26.9	0.0	NaN	NaN	N	
145458	2017-06-24	Uluru	7.8	27.0	0.0	NaN	NaN	SE	

142193 rows × 24 columns

In [121]:

```

1 X_train_enc, y_train, X_test_enc, y_test, preprocessor = preprocess_fea
2     train_df,
3     test_df,
4     numeric_features + ["Rainfall_lag1"],
5     categorical_features,
6     drop_features,
7 )

```

In [122]:

```

1 lr_coef = score_lr_print_coeff(
2     preprocessor, train_df, y_train, test_df, y_test, X_train_enc
3 )

```

Train score: 0.85

Test score: 0.84

In [123]: 1 lr_coef.loc[["Rainfall", "Rainfall_lag1"]]

Out[123]:

	Coef
Rainfall	0.081052
Rainfall_lag1	0.008290

- Rainfall from today has a positive coefficient.

- Rainfall from yesterday has a positive but a smaller coefficient.
- If we didn't have rainfall from today feature, rainfall from yesterday feature would have received a bigger coefficient.

- We could also create a lagged version of the target.
- In fact, this dataset already has that built in! `RainToday` is the lagged version of the target `RainTomorrow`.
- We could also create lagged version of other features, or more lags

```
In [124]: 1 rain_df_modified = create_lag_feature(rain_df, "Rainfall", 1)
2 rain_df_modified = create_lag_feature(rain_df_modified, "Rainfall", 2)
3 rain_df_modified = create_lag_feature(rain_df_modified, "Rainfall", 3)
4 rain_df_modified = create_lag_feature(rain_df_modified, "Humidity3pm",
```

```
In [125]: 1 rain_df_modified[
2     [
3         "Date",
4         "Location",
5         "Rainfall",
6         "Rainfall_lag1",
7         "Rainfall_lag2",
8         "Rainfall_lag3",
9         "Humidity3pm",
10        "Humidity3pm_lag1",
11    ]
12 ].head(10)
```

Out[125]:

	Date	Location	Rainfall	Rainfall_lag1	Rainfall_lag2	Rainfall_lag3	Humidity3pm	Humidity3pm_lag1
0	2008-12-01	Albury	0.6	NaN	NaN	NaN	22.0	Na
1	2008-12-02	Albury	0.0	0.6	NaN	NaN	25.0	22
2	2008-12-03	Albury	0.0	0.0	0.6	NaN	30.0	25
3	2008-12-04	Albury	0.0	0.0	0.0	0.6	16.0	30
4	2008-12-05	Albury	1.0	0.0	0.0	0.0	33.0	16
5	2008-12-06	Albury	0.2	1.0	0.0	0.0	23.0	33
6	2008-12-07	Albury	0.0	0.2	1.0	0.0	19.0	23
7	2008-12-08	Albury	0.0	0.0	0.2	1.0	19.0	19
8	2008-12-09	Albury	0.0	0.0	0.0	0.2	9.0	19
9	2008-12-10	Albury	1.4	0.0	0.0	0.0	27.0	9

Note the pattern of `NaN` values.

```
In [126]: 1 train_df = rain_df_modified.query("Date <= 20150630")
2 test_df = rain_df_modified.query("Date > 20150630")
```

```
In [127]: 1 X_train_enc, y_train, X_test_enc, y_test, preprocessor = preprocess_fea
2     train_df,
3     test_df,
4     numeric_features
5     + ["Rainfall_lag1", "Rainfall_lag2", "Rainfall_lag3", "Humidity3pm_"
6     categorical_features,
7     drop_features,
8     ]
```

```
In [128]: 1 lr_coef = score_lr_print_coeff(
2     preprocessor, train_df, y_train, test_df, y_test, X_train_enc
3     )
```

Train score: 0.85

Test score: 0.85

```
In [129]: 1 lr_coef.loc[
2     [
3         "Rainfall",
4         "Rainfall_lag1",
5         "Rainfall_lag2",
6         "Rainfall_lag3",
7         "Humidity3pm",
8         "Humidity3pm_lag1",
9     ]
10 ]
```

Out[129]:

	Coef
Rainfall	0.108519
Rainfall_lag1	0.023086
Rainfall_lag2	0.018295
Rainfall_lag3	0.017829
Humidity3pm	1.278602
Humidity3pm_lag1	-0.267480

Note the pattern in the magnitude of the coefficients.

```
In [ ]:
```

Forecasting further into the future

- Let's say we want to predict 7 days into the future instead of one day.
- There are a few main approaches here:

1. Train a separate model for each number of days. E.g. one model that predicts RainTomorrow, another model that predicts RainIn2Days, etc. We can build these datasets.
2. Use a multi-output model that jointly predicts RainTomorrow, RainIn2Days, etc. However, multi-output models are outside the scope of CPSC 330.
3. Use one model and sequentially predict using a `for` loop. However, this requires predicting *all* features into a model so may not be that useful here.

- To briefly dig into approach 3, this is easier to understand for a univariate (one feature) time series.
- To dig into this we'll look at the [Retail Sales of Clothing and Clothing Accessory Stores dataset \(<https://fred.stlouisfed.org/series/MRTSSM448USN>\)](https://fred.stlouisfed.org/series/MRTSSM448USN) made available by the Federal Reserve Bank of St. Louis.

```
In [130]: 1 retail_df = pd.read_csv("../data/MRTSSM448USN.csv", parse_dates=[ "DATE" ] )
2 retail_df.columns = [ "date", "sales" ]
```

```
In [131]: 1 retail_df.head()
```

```
Out[131]:      date  sales
0  1992-01-01   6938
1  1992-02-01   7524
2  1992-03-01   8475
3  1992-04-01   9401
4  1992-05-01   9558
```

```
In [132]: 1 retail_df[ "date" ].min()
```

```
Out[132]: Timestamp('1992-01-01 00:00:00')
```

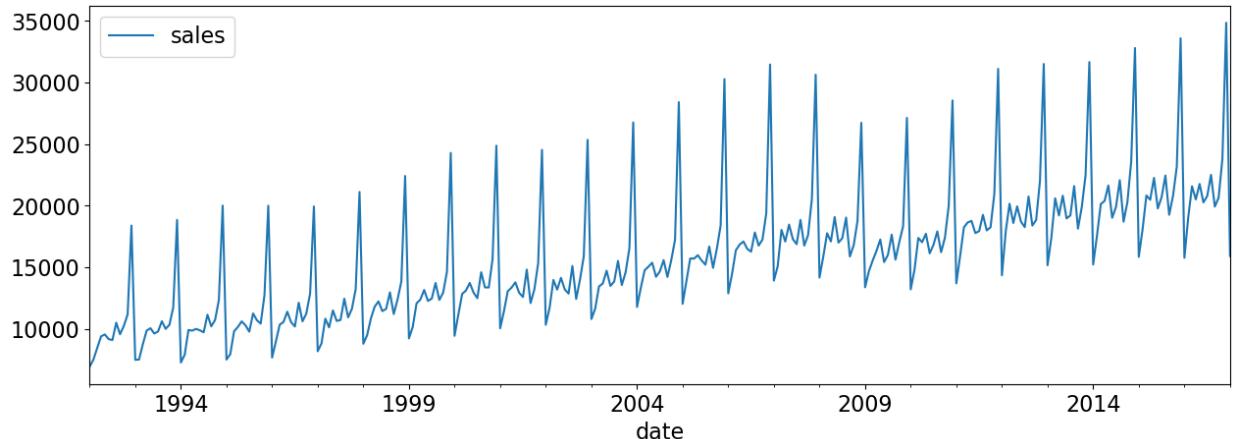
```
In [133]: 1 retail_df[ "date" ].max()
```

```
Out[133]: Timestamp('2023-01-01 00:00:00')
```

```
In [134]: 1 retail_df_train = retail_df.query("date <= 20170101")
2 retail_df_test = retail_df.query("date > 20170101")
```

In [135]:

```
1 retail_df_train.plot(x="date", y="sales", figsize=(15, 5));
```



We can create a dataset using purely lag features.

In [136]:

```
1 def lag_df(df, lag, cols):
2     return df.assign(
3         **{f"{col}-lag{n}": df[col].shift(n) for n in range(1, lag + 1) for col in cols})
4     )
```

In [137]:

```
1 retail_lag_5 = lag_df(retail_df, 5, ["sales"])
2 retail_train_5 = retail_lag_5.query("date <= 20170101")
3 retail_test_5 = retail_lag_5.query("date > 20170101")
4 retail_train_5
```

Out[137]:

	date	sales	sales-1	sales-2	sales-3	sales-4	sales-5
0	1992-01-01	6938	NaN	NaN	NaN	NaN	NaN
1	1992-02-01	7524	6938.0	NaN	NaN	NaN	NaN
2	1992-03-01	8475	7524.0	6938.0	NaN	NaN	NaN
3	1992-04-01	9401	8475.0	7524.0	6938.0	NaN	NaN
4	1992-05-01	9558	9401.0	8475.0	7524.0	6938.0	NaN
...
296	2016-09-01	19928	22505.0	20782.0	20274.0	21774.0	20514.0
297	2016-10-01	20650	19928.0	22505.0	20782.0	20274.0	21774.0
298	2016-11-01	23826	20650.0	19928.0	22505.0	20782.0	20274.0
299	2016-12-01	34847	23826.0	20650.0	19928.0	22505.0	20782.0
300	2017-01-01	15921	34847.0	23826.0	20650.0	19928.0	22505.0

301 rows × 7 columns

- Now, if we drop the "date" column we have a target ("sales") and 5 features (the previous 5 days of sales).
- We need to impute/drop the missing values and then we can fit a model to this. I will just drop for convenience:

```
In [138]: 1 retail_train_5 = retail_train_5[5: ].drop(columns=["date"])
2 retail_train_5
```

Out[138]:

	sales	sales-1	sales-2	sales-3	sales-4	sales-5
5	9182	9558.0	9401.0	8475.0	7524.0	6938.0
6	9103	9182.0	9558.0	9401.0	8475.0	7524.0
7	10513	9103.0	9182.0	9558.0	9401.0	8475.0
8	9573	10513.0	9103.0	9182.0	9558.0	9401.0
9	10254	9573.0	10513.0	9103.0	9182.0	9558.0
...
296	19928	22505.0	20782.0	20274.0	21774.0	20514.0
297	20650	19928.0	22505.0	20782.0	20274.0	21774.0
298	23826	20650.0	19928.0	22505.0	20782.0	20274.0
299	34847	23826.0	20650.0	19928.0	22505.0	20782.0
300	15921	34847.0	23826.0	20650.0	19928.0	22505.0

296 rows × 6 columns

```
In [139]: 1 retail_train_5_X = retail_train_5.drop(columns=["sales"])
2 retail_train_5_y = retail_train_5["sales"]
```

```
In [140]: 1 from sklearn.ensemble import RandomForestRegressor
```

```
In [141]: 1 retail_model = RandomForestRegressor()
2 retail_model.fit(retail_train_5_X, retail_train_5_y);
```

Given this, we can now predict the sales

```
In [142]: 1 preds = retail_model.predict(retail_test_5.drop(columns=["date", "sales"])
2 preds
```

Out[142]: array([18768.09, 20491.71, 20789.32, 22529.24, 20617. , 21838.07,
 22218.1 , 22352.96, 20851.81, 22713. , 31629.71, 15989.66,
 18587.88, 21489.34, 22392.5 , 21835.42, 28448.45, 20629.66,
 22276.94, 29463.9 , 21449.94, 22054.79, 21093.12, 15949.61,
 18826.44, 20505.58, 22669.61, 22053.39, 27977.7 , 22124.38,
 22006.33, 30543.09, 21478.95, 21951.6 , 28344.54, 15952.53,
 19153.75, 22005.46, 13647.61, 11574.87, 11231.21, 15020.43,
 12481.92, 11072.85, 18325.01, 20567.68, 22414.79, 14965.93,
 17696.88, 18930.01, 31319.99, 29013.66, 15351.69, 29359.58,
 26889.1 , 28723.33, 22943.54, 30084.94, 16856.13, 17222.57,
 20365.81, 22563.73, 19718.08, 17245.92, 17350.4 , 29038.05,
 26889.1 , 18079.17, 30265.52, 28723.33, 16856.13, 17222.57])

In [143]:

```

1 retail_test_5_preds = retail_test_5.assign(predicted_sales=preds)
2 retail_test_5_preds.head()

```

Out[143]:

		date	sales	sales-1	sales-2	sales-3	sales-4	sales-5	predicted_sales
301		2017-02-01	18036	15921.0	34847.0	23826.0	20650.0	19928.0	18768.09
302		2017-03-01	21348	18036.0	15921.0	34847.0	23826.0	20650.0	20491.71
303		2017-04-01	21154	21348.0	18036.0	15921.0	34847.0	23826.0	20789.32
304		2017-05-01	21954	21154.0	21348.0	18036.0	15921.0	34847.0	22529.24
305		2017-06-01	20623	21954.0	21154.0	21348.0	18036.0	15921.0	20617.00

- Ok, that is fine, but what if we want to predict 7 days in the future?
- Well, we would not have access to our features!! We don't yet know the previous day's sales, or 2 days prior!
- So we can use "Approach 3" mentioned earlier: predict these values and then pretend they are true!
- For simplicity, say today is Monday
 1. Predict Tuesday's sales
 2. Then, to predict for Wednesday, we need to know Tuesday's sales. Use our *prediction* for Tuesday as the truth.
 3. Then, to predict for Thursday, we need to know Tue and Wed sales. Use our predictions.
 4. Etc etc.

In []:

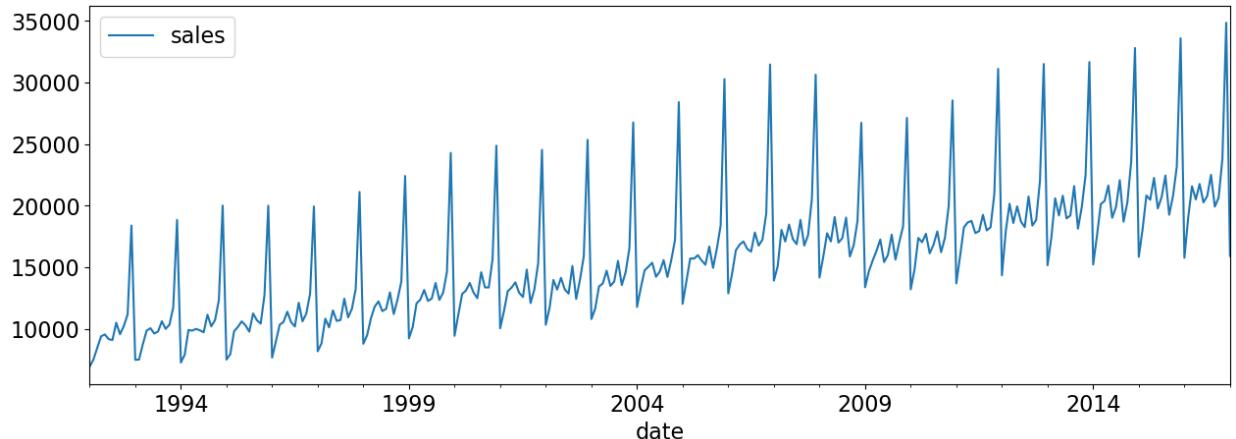
1

Trends

- There are some important concepts in time series that rely on having a continuous target (like we do in the retail sales example above).
- Part of that is the idea of seasonality and trends.
- These are mostly taken care of by our feature engineering of the data variable, but there's something important left to discuss.

In [144]:

```
1 retail_df_train.plot(x="date", y="sales", figsize=(15, 5));
```



- It looks like there's a **trend** here - the sales are going up over time.

Let's say we encoded the date as a feature in days like this:

In [145]:

```
1 retail_train_5_date = retail_lag_5.query("date <= 20170101")
2 first_day_retail = retail_train_5_date["date"].min()
3
4 retail_train_5_date = retail_train_5_date.assign(
5     Days_since=retail_train_5_date["date"].apply(lambda x: (x - first_d
6 )
7 retail_train_5_date.head(10)
```

Out[145]:

	date	sales	sales-1	sales-2	sales-3	sales-4	sales-5	Days_since
0	1992-01-01	6938	NaN	NaN	NaN	NaN	NaN	0
1	1992-02-01	7524	6938.0	NaN	NaN	NaN	NaN	31
2	1992-03-01	8475	7524.0	6938.0	NaN	NaN	NaN	60
3	1992-04-01	9401	8475.0	7524.0	6938.0	NaN	NaN	91
4	1992-05-01	9558	9401.0	8475.0	7524.0	6938.0	NaN	121
5	1992-06-01	9182	9558.0	9401.0	8475.0	7524.0	6938.0	152
6	1992-07-01	9103	9182.0	9558.0	9401.0	8475.0	7524.0	182
7	1992-08-01	10513	9103.0	9182.0	9558.0	9401.0	8475.0	213
8	1992-09-01	9573	10513.0	9103.0	9182.0	9558.0	9401.0	244
9	1992-10-01	10254	9573.0	10513.0	9103.0	9182.0	9558.0	274

- Now, let's say we use all these features (the lagged version of the target and also `Days_since`).
- If we use **linear regression** we'll learn a coefficient for `Days_since`.
 - If that coefficient is positive, it predicts unlimited growth forever. That may not be what you want? It depends.

- If we use a **random forest**, we'll just be doing splits from the training set, e.g. "if `Days_since > 9100` then do this".
 - There will be no splits for later time points because there is no training data there.
 - Thus tree-based models cannot model trends.
 - This is really important to know!!
- Often, we model the trend separately and use the random forest to model a de-trended time series

In []:

1

What did we not cover? (5 min)

- A huge amount!

Traditional time series approaches

- Time series analysis is a huge field of its own (notice a pattern here?)
- Traditional approaches include the [ARIMA model](https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average) (https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average) and its various components/extensions.
- In Python, the [statsmodels](https://www.statsmodels.org/) (<https://www.statsmodels.org/>) package is the place to go for this sort of thing.
 - For example, [statsmodels.tsa.arima_model.ARIMA](https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima_model.ARIMA.html) (https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima_model.ARIMA.html).
- These approaches can forecast, but they are also very good for understanding the temporal relationships in your data.
- We will take a different route in this course, and stick to our supervised learning tools.

Deep learning

- Recently, deep learning has been very successful too.
- In particular, [recurrent neural networks](https://en.wikipedia.org/wiki/Recurrent_neural_network) (https://en.wikipedia.org/wiki/Recurrent_neural_network) (RNNs).
 - These are not covered in CPSC 340, but I believe they are in 440.
 - [LSTMs](https://en.wikipedia.org/wiki/Long_short-term_memory) (https://en.wikipedia.org/wiki/Long_short-term_memory) especially have shown a lot of promise in this type of task.
 - [Here](https://colah.github.io/posts/2015-08-Understanding-LSTMs/) (<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>) is a blog post about LSTMs.

Types of problems involving time series

- A single label associated with an entire time series.
 - We had that with images earlier on, you could have the same for a time series.
 - E.g., for fraud detection, labelling each transaction as fraud/normal vs. labelling a person as bad/good based on their entire history.

- There are various approaches that can be used for this type of problem, including CNNs, LSTMs, and non deep learning methods.
- Inference problems.
 - What are the patterns in this time series?
 - How many lags are associated with the current value?
- Etc.

Unequally spaced time points

- We assumed we have a measurement each day.
- For example, when creating lag features we used consecutive rows in the DataFrame.
- But, in fact some days were missing in this dataset.
- More generally, what if the measurements are at arbitrary times, not equally spaced?
 - Some of our approaches would still work, like encoding the month / looking at seasonality.
 - Some of our approaches would not make sense, like the lags.
 - Perhaps the measurements could be binned into equally spaced bins, or something.
 - This is more of a hassle.

Other software package

- One good one to know about is [Prophet](https://facebook.github.io/prophet/docs/quick_start.html) (https://facebook.github.io/prophet/docs/quick_start.html).

Feature engineering

- Often, a useful approach is to just *engineer your own features*.
 - E.g., max expenditure, min expenditure, max-min, avg time gap between transactions, variance of time gap between transactions, etc etc.
 - We could do that here as well, or in any problem.

CPSC 330

Applied Machine Learning

Lecture 20: Survival analysis

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

```
In [1]: 1 import matplotlib.pyplot as plt
2 import numpy as np
3 import pandas as pd
4 from sklearn.compose import ColumnTransformer, make_column_transformer
5 from sklearn.dummy import DummyClassifier
6 from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
7 from sklearn.impute import SimpleImputer
8 from sklearn.linear_model import LogisticRegression, Ridge
9 from sklearn.metrics import confusion_matrix, plot_confusion_matrix
10 from sklearn.model_selection import (
11     cross_val_predict,
12     cross_val_score,
13     cross_validate,
14     train_test_split,
15 )
16 from sklearn.pipeline import Pipeline, make_pipeline
17 from sklearn.preprocessing import (
18     FunctionTransformer,
19     OneHotEncoder,
20     OrdinalEncoder,
21     StandardScaler,
22 )
23
24 plt.rcParams["font.size"] = 16
25
26 # does lifelines try to mess with this?
27 pd.options.display.max_rows = 10
```

```
In [2]: 1 import lifelines
```

Learning objectives

- Explain the problem with treating right-censored data the same as "regular" data.

- Determine whether survival analysis is an appropriate tool for a given problem.
- Apply survival analysis in Python using the `lifelines` package.
- Interpret a survival curve, such as the Kaplan-Meier curve.
- Interpret the coefficients of a fitted Cox proportional hazards model.
- Make predictions for existing individuals and interpret these predictions.

Customer churn: our standard approach

- In hw5 you looked at a dataset about [customer churn](https://en.wikipedia.org/wiki/Customer_attrition) (https://en.wikipedia.org/wiki/Customer_attrition).
- In hw5, the dataset was interesting because it's unbalanced (most customers stay). We used typical binary classification approach on the dataset.
- Today we'll look at a different customer churn [dataset](https://www.kaggle.com/blastchar/telco-customer-churn) (<https://www.kaggle.com/blastchar/telco-customer-churn>), because it has a feature we need - time!
- We'll explore the time aspect of the dataset today.

```
In [5]: 1 df = pd.read_csv("../data/WA_Fn-UseC_-Telco-Customer-Churn.csv")
2 train_df, test_df = train_test_split(df, random_state=123)
3 train_df.head()
```

Out[5]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
6464	4726-DLWQN	Male	1	No	No	50	Yes	Yes
5707	4537-DKTAL	Female	0	No	No	2	Yes	No
3442	0468-YRPXN	Male	0	No	No	29	Yes	No
3932	1304-NECVQ	Female	1	No	No	2	Yes	Yes
6124	7153-CHRBV	Female	0	Yes	Yes	57	Yes	No

5 rows × 21 columns

We can treat this as a binary classification problem where we want to predict Churn (yes/no) from these other columns.

```
In [6]: 1 train_df.shape
```

Out[6]: (5282, 21)

```
In [7]: 1 train_df["Churn"].value_counts()
```

Out[7]: No 3912
Yes 1370
Name: Churn, dtype: int64

```
In [8]: 1 train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5282 entries, 6464 to 3582
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   customerID      5282 non-null    object  
 1   gender          5282 non-null    object  
 2   SeniorCitizen   5282 non-null    int64  
 3   Partner         5282 non-null    object  
 4   Dependents     5282 non-null    object  
 5   tenure          5282 non-null    int64  
 6   PhoneService    5282 non-null    object  
 7   MultipleLines   5282 non-null    object  
 8   InternetService 5282 non-null    object  
 9   OnlineSecurity  5282 non-null    object  
 10  OnlineBackup    5282 non-null    object  
 11  DeviceProtection 5282 non-null    object  
 12  TechSupport    5282 non-null    object  
 13  StreamingTV    5282 non-null    object  
 14  StreamingMovies 5282 non-null    object  
 15  Contract        5282 non-null    object  
 16  PaperlessBilling 5282 non-null    object  
 17  PaymentMethod   5282 non-null    object  
 18  MonthlyCharges 5282 non-null    float64 
 19  TotalCharges   5282 non-null    object  
 20  Churn           5282 non-null    object  
dtypes: float64(1), int64(2), object(18)
memory usage: 907.8+ KB
```

Question: Does this mean there is no missing data?

Ok, let's try our usual approach:

```
In [9]: 1 train_df[ "SeniorCitizen" ].value_counts()
```

```
Out[9]: 0    4430
1    852
Name: SeniorCitizen, dtype: int64
```

```
In [10]: 1 numeric_features = ["tenure", "MonthlyCharges", "TotalCharges"]
2 drop_features = ["customerID"]
3 passthrough_features = ["SeniorCitizen"]
4 target_column = ["Churn"]
5 # the rest are categorical
6 categorical_features = list(
7     set(train_df.columns)
8     - set(numeric_features)
9     - set(passthrough_features)
10    - set(drop_features)
11    - set(target_column)
12 )
```

```
In [11]: 1 preprocessor = make_column_transformer(
2     (StandardScaler(), numeric_features),
3     (OneHotEncoder(), categorical_features),
4     ("passthrough", passthrough_features),
5     ("drop", drop_features),
6 )
```

```
In [12]: 1 preprocessor.fit(train_df);
```

```
-----
--> ValueError                                     Traceback (most recent call last)
Cell In[12], line 1
----> 1 preprocessor.fit(train_df);

File /opt/anaconda3/envs/cpsc330/lib/python3.10/site-packages/sklearn/compose/_column_transformer.py:657, in ColumnTransformer.fit(self, X, y)
 639 """Fit all transformers using X.
 640
 641 Parameters
 642     (...)
 653     This estimator.
 654 """
 655 # we use fit_transform to make sure to set sparse_output_ (for which we
 656 # need the transformed data) to have consistent output type in predict
 657
 658     ...
```

Hmmm, one of the numeric features is causing problems?

In [13]:

```
1 df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   customerID      7043 non-null    object  
 1   gender          7043 non-null    object  
 2   SeniorCitizen   7043 non-null    int64  
 3   Partner         7043 non-null    object  
 4   Dependents     7043 non-null    object  
 5   tenure          7043 non-null    int64  
 6   PhoneService    7043 non-null    object  
 7   MultipleLines   7043 non-null    object  
 8   InternetService 7043 non-null    object  
 9   OnlineSecurity  7043 non-null    object  
 10  OnlineBackup    7043 non-null    object  
 11  DeviceProtection 7043 non-null    object  
 12  TechSupport    7043 non-null    object  
 13  StreamingTV    7043 non-null    object  
 14  StreamingMovies 7043 non-null    object  
 15  Contract        7043 non-null    object  
 16  PaperlessBilling 7043 non-null    object  
 17  PaymentMethod   7043 non-null    object  
 18  MonthlyCharges 7043 non-null    float64 
 19  TotalCharges    7043 non-null    object  
 20  Churn           7043 non-null    object  
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

Oh, looks like `TotalCharges` is not a numeric type. What if we change the type of this column to float?

```
In [14]: 1 train_df["TotalCharges"] = train_df["TotalCharges"].astype(float)
```

```

-----
--> 1     raise ValueError("TotalCharges must be float")
          ^
ValueError: TotalCharges must be float
-----
```

Traceback (most recent call last)

t)
Cell In[14], line 1
----> 1 train_df["TotalCharges"] = train_df["TotalCharges"].astype(float)

File /opt/anaconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/core/generic.py:6240, in NDFrame.astype(self, dtype, copy, errors)
 6233 results = [
 6234 self.iloc[:, i].astype(dtype, copy=copy)
 6235 for i in range(len(self.columns))
 6236]
 6238 else:
 6239 # else, only a single dtype is given
-> 6240 new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors)
 6241 return self._constructor(new_data).__finalize__(self, method="astype")
 6243 # GH 33113: handle empty frame or series

File /opt/anaconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/core/internals/managers.py:448, in BaseBlockManager.astype(self, dtype, copy, errors)
 447 def astype(self: T, dtype, copy: bool = False, errors: str = "raise") -> T:
--> 448 return self.apply("astype", dtype=dtype, copy=copy, errors=errors)

File /opt/anaconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/core/internals/managers.py:352, in BaseBlockManager.apply(self, f, align_keys, ignore_failures, **kwargs)
 350 applied = b.apply(f, **kwargs)
 351 else:
--> 352 applied = getattr(b, f)(**kwargs)
 353 except (TypeError, NotImplementedError):
 354 if not ignore_failures:

File /opt/anaconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/core/internals/blocks.py:526, in Block.astype(self, dtype, copy, errors)
 508 """
 509 Coerce to the new dtype.
 510
(...)
 522 Block
 523 """
 524 values = self.values
--> 526 new_values = astype_array_safe(values, dtype, copy=copy, errors=errors)
 528 new_values = maybe_coerce_values(new_values)
 529 newb = self.make_block(new_values)

File /opt/anaconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/core/dtypes/astype.py:299, in astype_array_safe(values, dtype, copy, errors)
 296 return values.copy()
 298 try:
--> 299 new_values = astype_array(values, dtype, copy=copy)

```

300 except (ValueError, TypeError):
301     # e.g. astype_nansafe can fail on object-dtype of strings
302     # trying to convert to float
303     if errors == "ignore":

File /opt/anaconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/cor
e/dtypes/astype.py:230, in astype_array(values, dtype, copy)
227     values = values.astype(dtype, copy=copy)
229 else:
--> 230     values = astype_nansafe(values, dtype, copy=copy)
232 # in pandas we don't store numpy str dtypes, so convert to object
233 if isinstance(dtype, np.dtype) and issubclass(values.dtype.type,
str):

File /opt/anaconda3/envs/cpsc330/lib/python3.10/site-packages/pandas/cor
e/dtypes/astype.py:170, in astype_nansafe(arr, dtype, copy, skipna)
166     raise ValueError(msg)
168 if copy or is_object_dtype(arr.dtype) or is_object_dtype(dtype):
169     # Explicit copy, or required since NumPy can't view from / to
object.
--> 170     return arr.astype(dtype, copy=True)
172 return arr.astype(dtype, copy=copy)

ValueError: could not convert string to float: ''

```

Argh!!

In [15]:

```

1 for val in train_df["TotalCharges"]:
2     try:
3         float(val)
4     except ValueError:
5         print(val)

```

Any ideas?

Well, it turns out we can't see those problematic values because they are whitespace!

```
In [16]: 1 for val in train_df["TotalCharges"]:
2     try:
3         float(val)
4     except ValueError:
5         print('"%s"' % val)

" "
" "
" "
" "
" "
" "
" "
" "
```

Let's replace the whitespaces with NaNs.

```
In [17]: 1 train_df = train_df.assign(
2     TotalCharges=train_df["TotalCharges"].replace(" ", np.nan).astype(f
3 )
4 test_df = test_df.assign(
5     TotalCharges=test_df["TotalCharges"].replace(" ", np.nan).astype(fl
6 )
```

```
In [18]: 1 train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5282 entries, 6464 to 3582
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
---  --  
 0   customerID      5282 non-null    object 
 1   gender          5282 non-null    object 
 2   SeniorCitizen   5282 non-null    int64  
 3   Partner         5282 non-null    object 
 4   Dependents     5282 non-null    object 
 5   tenure          5282 non-null    int64  
 6   PhoneService    5282 non-null    object 
 7   MultipleLines   5282 non-null    object 
 8   InternetService 5282 non-null    object 
 9   OnlineSecurity  5282 non-null    object 
 10  OnlineBackup    5282 non-null    object 
 11  DeviceProtection 5282 non-null    object 
 12  TechSupport    5282 non-null    object 
 13  StreamingTV    5282 non-null    object 
 14  StreamingMovies 5282 non-null    object 
 15  Contract        5282 non-null    object 
 16  PaperlessBilling 5282 non-null    object 
 17  PaymentMethod   5282 non-null    object 
 18  MonthlyCharges  5282 non-null    float64
 19  TotalCharges    5274 non-null    float64
 20  Churn           5282 non-null    object 
dtypes: float64(2), int64(2), object(17)
memory usage: 907.8+ KB
```

But now we are going to have missing values and we need to include imputation for numeric features in our preprocessor.

```
In [19]: 1  preprocessor = make_column_transformer(
2      (
3          make_pipeline(SimpleImputer(strategy="median"), StandardScaler(
4              numeric_features,
5          )),
6          (OneHotEncoder(handle_unknown="ignore"), categorical_features),
7          ("passthrough", passthrough_features),
8          ("drop", drop_features),
9      )
```

Now let's try that again...

```
In [20]: 1  preprocessor.fit(train_df);
```

It worked! Let's get the column names of the transformed data from the column transformer.

```
In [22]: 1  new_columns = (
2      numeric_features
3      + preprocessor.named_transformers_["onehotencoder"]
4      .get_feature_names_out(categorical_features)
5      .tolist()
6      + passthrough_features
7  )
```

```
In [23]: 1  X_train_enc = pd.DataFrame(
2      preprocessor.transform(train_df), index=train_df.index, columns=new
3  )
4  X_test_enc = pd.DataFrame(
5      preprocessor.transform(test_df), index=test_df.index, columns=new
6  )
```

```
In [24]: 1  X_train_enc.head()
```

	tenure	MonthlyCharges	TotalCharges	TechSupport_No	TechSupport_No internet service	TechSupport_Yes
6464	0.707712	0.185175	0.513678	1.0	0.0	0.0
5707	-1.248999	-0.641538	-0.979562	1.0	0.0	0.0
3442	-0.148349	1.133562	0.226789	0.0	0.0	1.0
3932	-1.248999	0.458524	-0.950696	1.0	0.0	0.0
6124	0.993065	-0.183179	0.433814	0.0	0.0	1.0

5 rows × 45 columns

Before we look into survival analysis, let's just treat it as a binary classification model where we want to predict whether a customer churned or not.

In [25]: 1 results = {}

In [26]: 1 **def** mean_std_cross_val_scores(model, X_train, y_train, **kwargs):
2 """
3 Returns mean and std of cross validation
4
5 Parameters
6 -----
7 model :
8 scikit-learn model
9 X_train : numpy array or pandas DataFrame
10 X in the training data
11 y_train :
12 y in the training data
13
14 Returns
15 -----
16 pandas Series with mean scores from cross_validation
17 """
18
19 scores = cross_validate(model, X_train, y_train, **kwargs)
20
21 mean_scores = pd.DataFrame(scores).mean()
22 std_scores = pd.DataFrame(scores).std()
23 out_col = []
24
25 **for** i **in** range(len(mean_scores)):
26 out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))
27
28 **return** pd.Series(data=out_col, index=mean_scores.index)

In [27]: 1 X_train = train_df.drop(columns=["Churn"])
2 X_test = test_df.drop(columns=["Churn"])
3
4 y_train = train_df["Churn"]
5 y_test = test_df["Churn"]

DummyClassifier

In [28]: 1 dc = DummyClassifier()

```
In [29]: 1 results[ "dummy" ] = mean_std_cross_val_scores(
2         dc, X_train, y_train, return_train_score=True
3     )
4 pd.DataFrame(results)
```

Out[29]:

	dummy
fit_time	0.003 (+/- 0.001)
score_time	0.001 (+/- 0.000)
test_score	0.741 (+/- 0.000)
train_score	0.741 (+/- 0.000)

LogisticRegression

```
In [30]: 1 lr = make_pipeline(preprocessor, LogisticRegression(max_iter=1000))
```

```
In [31]: 1 results[ "logistic regression" ] = mean_std_cross_val_scores(
2         lr, X_train, y_train, return_train_score=True
3     )
4 pd.DataFrame(results)
```

Out[31]:

	dummy	logistic regression
fit_time	0.003 (+/- 0.001)	0.055 (+/- 0.008)
score_time	0.001 (+/- 0.000)	0.006 (+/- 0.000)
test_score	0.741 (+/- 0.000)	0.804 (+/- 0.013)
train_score	0.741 (+/- 0.000)	0.809 (+/- 0.002)

```
In [32]: 1 confusion_matrix(y_train, cross_val_predict(lr, X_train, y_train))
```

Out[32]: array([[3516, 396],
 [637, 733]])

RandomForestClassifier

```
In [33]: 1 rf = make_pipeline(preprocessor, RandomForestClassifier())
```

```
In [34]: 1 results[ "random forest" ] = mean_std_cross_val_scores(
2         rf, X_train, y_train, return_train_score=True
3     )
4 pd.DataFrame(results)
```

Out[34]:

	dummy	logistic regression	random forest
fit_time	0.003 (+/- 0.001)	0.055 (+/- 0.008)	0.287 (+/- 0.014)
score_time	0.001 (+/- 0.000)	0.006 (+/- 0.000)	0.020 (+/- 0.001)
test_score	0.741 (+/- 0.000)	0.804 (+/- 0.013)	0.790 (+/- 0.012)
train_score	0.741 (+/- 0.000)	0.809 (+/- 0.002)	0.998 (+/- 0.000)

```
In [35]: 1 confusion_matrix(y_train, cross_val_predict(rf, X_train, y_train))
```

Out[35]: array([[3518, 394],
 [735, 635]])

- This is what we did in hw5.
- What's wrong with this approach?

And now the rest of the class is about what is wrong with what we just did!

Censoring and survival analysis

Time to event and censoring

Imagine that you want to analyze *the time until an event occurs*. For example,

- the time until a disease kills its host.
- the time until a piece of equipment breaks.
- the time that someone unemployed will take to land a new job.
- the time until a customer leaves a subscription service (this dataset).

In our example, instead of predicting the binary label churn or no churn, it will be more useful to estimate when the customer is likely to churn (the time until churn happens) so that we can take some action.

In [36]: 1 train_df[["tenure"]].head()

Out[36]:

tenure	
6464	50
5707	2
3442	29
3932	2
6124	57

The tenure column is the number of months the customer has stayed with the company.

Although this branch of statistics is usually referred to as **Survival Analysis**, the event in question does not need to be related to actual "survival". The important thing is to understand that we are interested in **the time until something happens**, or whether or not something will happen in a certain time frame.

Question: But why is this different? Can't you just use the techniques you learned so far (e.g., regression models) to predict the time? Take a minute to think about this.

The answer would be yes if you could observe the actual time in all occurrences, but you usually cannot. Frequently, there will be some kind of **censoring** which will not allow you to observe the exact time that the event happened for all units/individuals that are being studied.

In [37]: 1 train_df[["tenure", "Churn"]].head()

Out[37]:

	tenure	Churn
6464	50	No
5707	2	No
3442	29	No
3932	2	Yes
6124	57	No

- What this means is that we **don't have correct target values** to train or test our model.
- This is a problem!

Let's consider some approaches to deal with this censoring issue.

Approach 1: Only consider the examples where "Churn"=Yes

Let's just consider the cases *for which we have the time*, to obtain the average subscription length.

```
In [38]: 1 train_df_churn = train_df.query(
2     "Churn == 'Yes'"
3 ) # Consider only examples where the customers churned.
4 test_df_churn = test_df.query(
5     "Churn == 'Yes'"
6 ) # Consider only examples where the customers churned.
7 train_df_churn.head()
```

Out[38]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	
3932	1304-NECVQ	Female		1	No	No	2	Yes	Yes
301	8098-LLAZX	Female		1	No	No	4	Yes	Yes
5540	3803-KMQFW	Female		0	Yes	Yes	1	Yes	No
4084	2777-PHDEI	Female		0	No	No	1	Yes	No
3272	6772-KSATR	Male		0	No	No	1	Yes	Yes

5 rows × 21 columns

```
In [39]: 1 train_df.shape
```

Out[39]: (5282, 21)

```
In [40]: 1 train_df_churn.shape
```

Out[40]: (1370, 21)

```
In [41]: 1 numeric_features
```

Out[41]: ['tenure', 'MonthlyCharges', 'TotalCharges']

```
In [42]: 1 preprocessing_notenure = make_column_transformer(
2     (
3         make_pipeline(SimpleImputer(strategy="median"), StandardScaler(
4             numeric_features[1:]), # Getting rid of the tenure column
5     ),
6         (OneHotEncoder(handle_unknown="ignore"), categorical_features),
7         ("passthrough", passthrough_features),
8 )
```

```
In [43]: 1 tenure_lm = make_pipeline(preprocessing_notenure, Ridge())
2
3 tenure_lm.fit(train_df_churn.drop(columns=["tenure"]), train_df_churn["
```

In [44]:

```
1 pd.DataFrame(  
2     tenure_lm.predict(test_df_churn.drop(columns=["tenure"]))[:10],  
3     columns=["tenure_predictions"],  
4 )
```

Out[44]:

tenure_predictions

	tenure_predictions
0	5.062449
1	13.198645
2	11.859455
3	5.865562
4	58.154842
5	3.757932
6	18.932070
7	7.720893
8	36.818041
9	7.263541

What will be wrong with our estimated survival times? Will they be too low or too high?

On average they will be **underestimates** (too small), because we are ignoring the currently subscribed (un-churned) customers. Our dataset is a biased sample of those who churned within the time window of the data collection. Long-time subscribers were more likely to be removed from the dataset! This is a common mistake - see the [Calling Bullshit video](#) (<https://www.youtube.com/watch?v=ITWQ5psx9Sw>) I posted on the README!

Approach 2: Assume everyone churns right now

Assume everyone churns right now - in other words, use the original dataset.

```
In [45]: 1 train_df[["tenure", "Churn"]].head()
```

Out[45]:

	tenure	Churn
6464	50	No
5707	2	No
3442	29	No
3932	2	Yes
6124	57	No

```
In [46]: 1 tenure_lm.fit(train_df.drop(columns=["tenure"]), train_df["tenure"]);
```

```
In [47]: 1 pd.DataFrame(  
2     tenure_lm.predict(test_df_churn.drop(columns=["tenure"])))[:10],  
3     columns=[ "tenure_predictions"],  
4 )
```

Out[47]:

	tenure_predictions
0	6.400047
1	20.220392
2	22.332746
3	12.825470
4	59.885968
5	7.075453
6	17.731498
7	10.407862
8	38.425365
9	10.854500

What will be wrong with our estimated survival time?

In [48]: 1 train_df[["tenure", "Churn"]].head()

Out[48]:

	tenure	Churn
6464	50	No
5707	2	No
3442	29	No
3932	2	Yes
6124	57	No

It will be an **underestimate** again. For those still subscribed, while we did not remove them, we recorded a total tenure shorter than in reality, because they will keep going for some amount of time.

Approach 3: Survival analysis

Deal with this properly using [survival analysis \(\[https://en.wikipedia.org/wiki/Survival_analysis\]\(https://en.wikipedia.org/wiki/Survival_analysis\)\)](https://en.wikipedia.org/wiki/Survival_analysis).

- You may learn about this in a statistics course.
- We will use the `lifelines` package in Python and will not go into the math/stats of how it works.

In [49]: 1 train_df[["tenure", "Churn"]].head()

Out[49]:

	tenure	Churn
6464	50	No
5707	2	No
3442	29	No
3932	2	Yes
6124	57	No

Types of questions we might want to answer:

1. How long do customers stay with the service?
2. For a particular customer, can we predict how long they might stay with the service?
3. What factors influence a customer's churn time?

Break (5 min)



Kaplan-Meier survival curve

Before we do anything further, I want to modify our dataset slightly:

1. I'm going to drop the `TotalCharges` (yes, after all that work fixing it) because it's a bit of a strange feature.
 - Its value actually changes over time, but we only have the value at the end.
 - We still have `MonthlyCharges`.
2. I'm going to not scale the `tenure` column, since it will be convenient to keep it in its original units of months.

Just for our sanity, I'm redefining the features.

In [50]:

```

1 numeric_features = ["MonthlyCharges"]
2 drop_features = ["customerID", "TotalCharges"]
3 passthrough_features = ["tenure", "SeniorCitizen"] # don't want to scale
4 target_column = ["Churn"]
5 # the rest are categorical
6 categorical_features = list(
7     set(train_df.columns)
8     - set(numeric_features)
9     - set(passthrough_features)
10    - set(drop_features)
11    - set(target_column)
12 )

```

In [51]:

```

1 preprocessing_final = make_column_transformer(
2     (
3         FunctionTransformer(lambda x: x == "Yes"),
4         target_column,
5     ), # because we need it in this format for lifelines package
6     ("passthrough", passthrough_features),
7     (StandardScaler(), numeric_features),
8     (OneHotEncoder(handle_unknown="ignore", sparse=False), categorical_
9      ("drop", drop_features)),
10 )

```

In [52]:

```
1 preprocessing_final.fit(train_df);
```

Let's get the column names of the columns created by our column transformer.

In [54]:

```

1 new_columns = (
2     target_column
3     + passthrough_features
4     + numeric_features
5     + preprocessing_final.named_transformers_["onehotencoder"]
6     .get_feature_names_out(categorical_features)
7     .tolist()
8 )

```

In [55]:

```

1 train_df_surv = pd.DataFrame(
2     preprocessing_final.transform(train_df), index=train_df.index, columns=new_columns
3 )
4 test_df_surv = pd.DataFrame(
5     preprocessing_final.transform(test_df), index=test_df.index, columns=new_columns
6 )

```

In [56]: 1 train_df_surv.head()

Out[56]:

	Churn	tenure	SeniorCitizen	MonthlyCharges	TechSupport_No	TechSupport_No internet service	TechSupport
6464	0.0	50.0	1.0	0.185175	1.0	0.0	0.0
5707	0.0	2.0	0.0	-0.641538	1.0	0.0	0.0
3442	0.0	29.0	0.0	1.133562	0.0	0.0	0.0
3932	1.0	2.0	1.0	0.458524	1.0	0.0	0.0
6124	0.0	57.0	0.0	-0.183179	0.0	0.0	0.0

5 rows × 45 columns

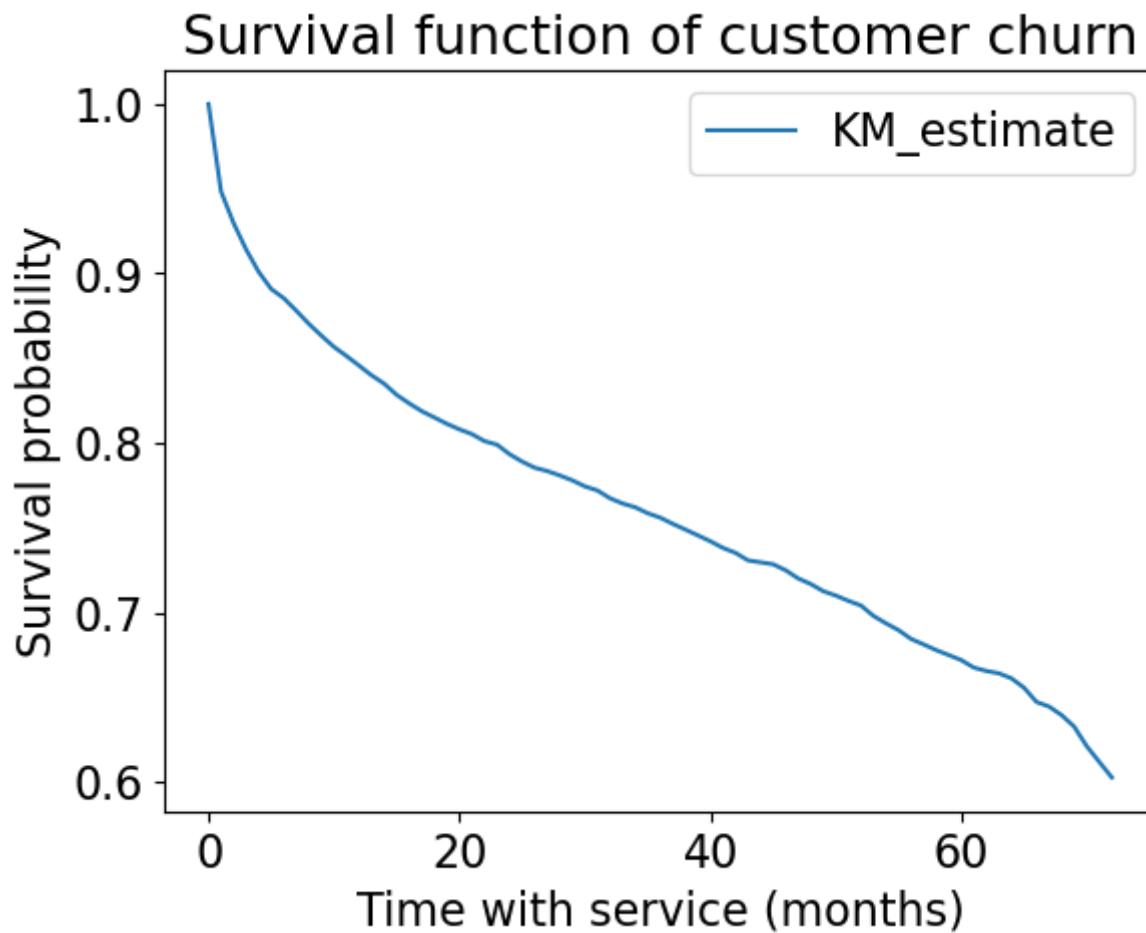
- We'll start with a model called `KaplanMeierFitter` from `lifelines` package to get a Kaplan Meier curve.
- For this model we only use two columns: tenure and churn.
- We do not use any other features.

In [57]: 1 kmf = lifelines.KaplanMeierFitter()

2 kmf.fit(train_df_surv["tenure"], train_df_surv["Churn"]);

In [58]:

```
1 kmf.survival_function_.plot()  
2 plt.title("Survival function of customer churn")  
3 plt.xlabel("Time with service (months)")  
4 plt.ylabel("Survival probability");
```



- What is this plot telling us?
- It shows the probability of survival over time.
- For example, after 20 months the probability of survival is ~0.8.
- Over time it's going down.

In [64]:

```
1 np.mean(y_train == "No")
```

Out[64]:

0.7406285497917455

What's the average tenure?

In [65]:

```
1 np.mean(train_df_surv["tenure"])
```

Out[65]:

32.6391518364256

What's the average tenure of the people who churned?

```
In [66]: 1 np.mean(train_df_surv.query("Churn == 1.0")["tenure"])
```

Out[66]: 17.854744525547446

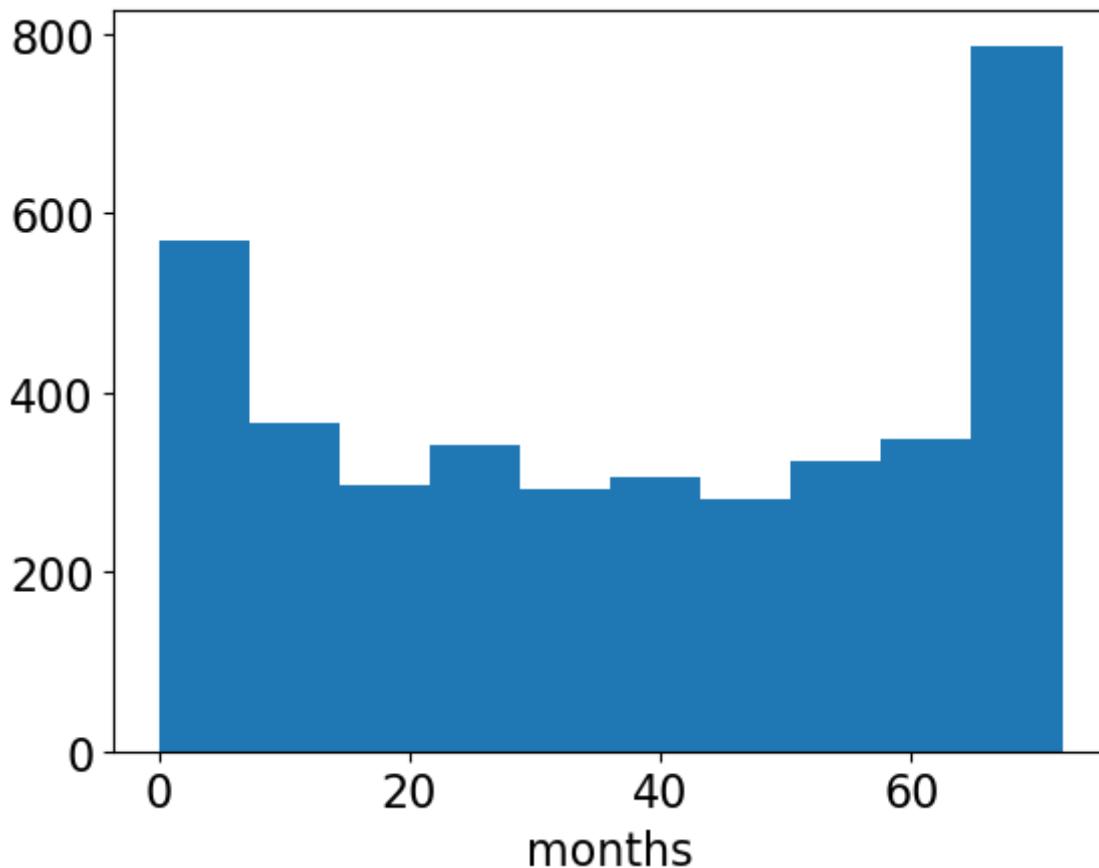
What's the average tenure of the people who did not churn?

```
In [67]: 1 np.mean(train_df_surv.query("Churn == 0.0")["tenure"])
```

Out[67]: 37.816717791411044

- Let's look at the histogram of number of people who have not churned.
- The key point here is that people *joined at different times*.

```
In [68]: 1 train_df[y_train == "No"]["tenure"].hist(grid=False)
2 plt.xlabel("months");
```



- Since the data was collected at a fixed time and these are the people who hadn't yet churned, those with larger `tenure` values here must have joined earlier.

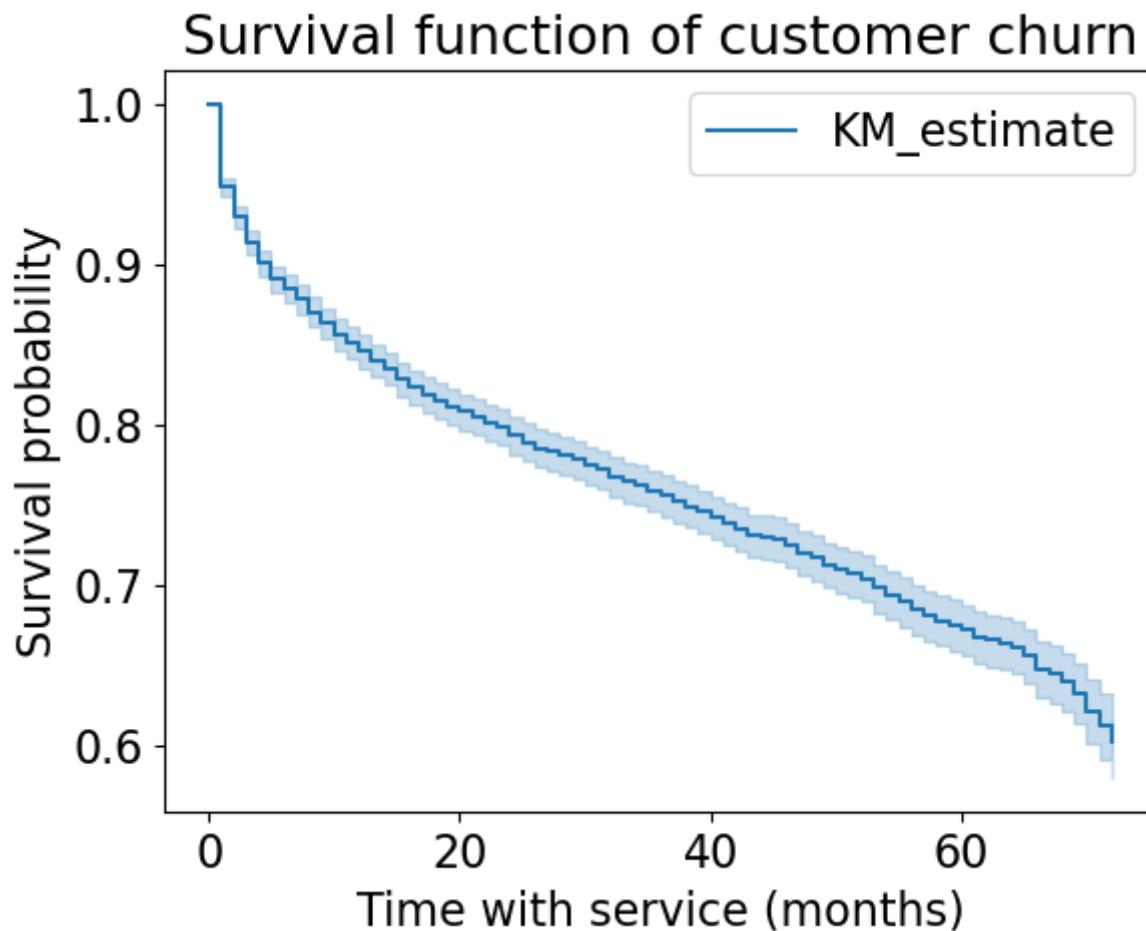
Lifelines can also give us some "error bars":

In [69]:

```

1 kmf.plot()
2 plt.title("Survival function of customer churn")
3 plt.xlabel("Time with service (months)")
4 plt.ylabel("Survival probability");

```



- We already have some actionable information here.
- The curve drops down fast at the beginning suggesting that people tend to leave early on.
- If there would have been a big drop in the curve, it means a bunch of people left at that time (e.g., after a 1-month free trial).
- BTW, the [original paper by Kaplan and Meier](https://web.stanford.edu/~lutian/coursepdf/KMpaper.pdf) (<https://web.stanford.edu/~lutian/coursepdf/KMpaper.pdf>) has been cited over 57000 times!

We can also create the K-M curve for different subgroups:

In [70]:

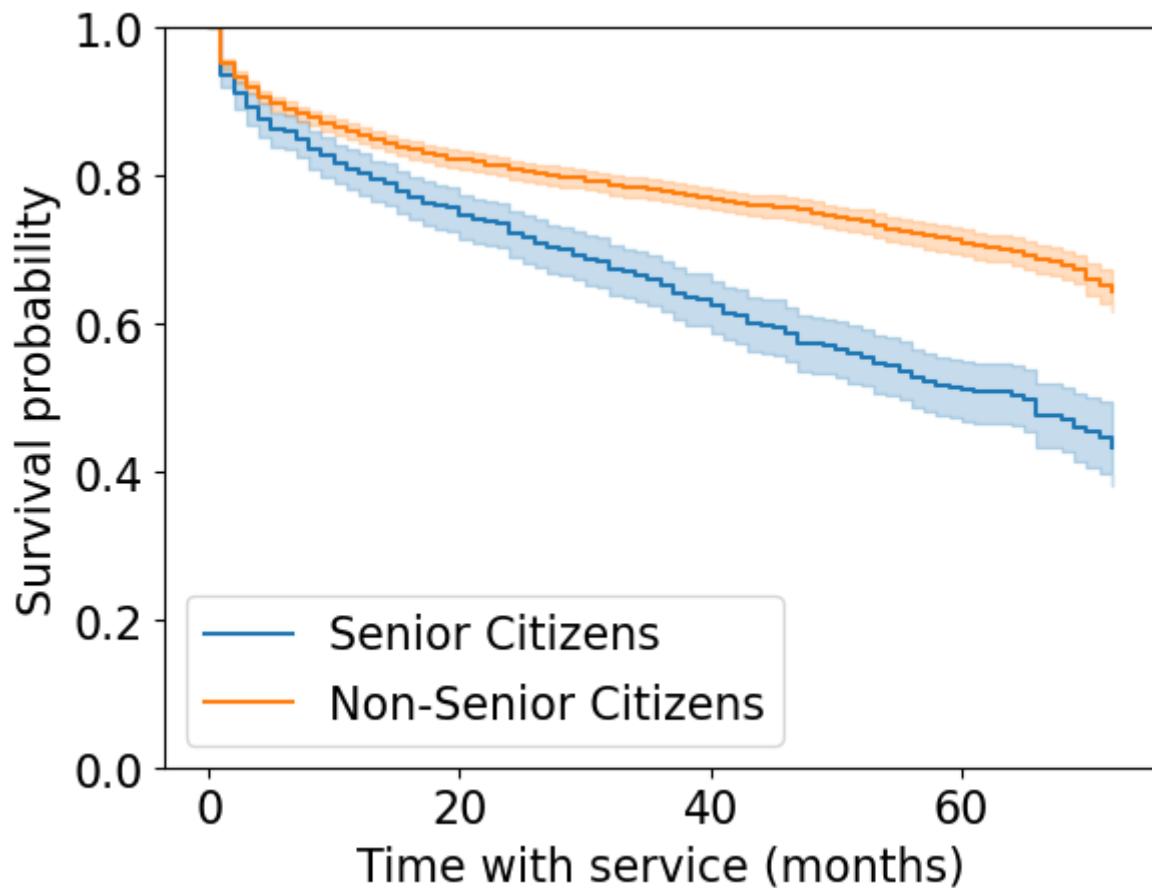
```

1 T = train_df_surv[ "tenure" ]
2 E = train_df_surv[ "Churn" ]
3 senior = train_df_surv[ "SeniorCitizen" ] == 1

```

In [71]:

```
1 ax = plt.subplot(111)
2
3 kmf.fit(T[senior], event_observed=E[senior], label="Senior Citizens")
4 kmf.plot(ax=ax)
5
6 kmf.fit(T[~senior], event_observed=E[~senior], label="Non-Senior Citizens")
7 kmf.plot(ax=ax)
8
9 plt.ylim(0, 1)
10 plt.xlabel("Time with service (months)")
11 plt.ylabel("Survival probability");
```



- It looks like senior citizens churn more quickly than others.
- This is quite useful!

Cox proportional hazards model

- We haven't been incorporating other features in the model so far.
- The Cox proportional hazards model is a commonly used model that allows us to interpret how features influence a censored tenure/duration.

- You can think of it like linear regression for survival analysis: we will get a coefficient for each feature that tells us how it influences survival.
- It makes some strong assumptions (the proportional hazards assumption) that may not be true, but we won't go into this here.
- The proportional hazard model works multiplicatively, like linear regression with log-transformed targets.

In [72]:

```

1 cph = lifelines.CoxPHFitter()
2 cph.fit(train_df_surv, duration_col="tenure", event_col="Churn");
-----
```

--

`LinAlgError` Traceback (most recent call last)

File /opt/anaconda3/envs/cpsc330/lib/python3.10/site-packages/lifelines/fitters/coxph_fitter.py:1527, in SemiParametricPHFitter._newton_raphson_for_efron_model(self, X, T, E, weights, entries, initial_point, show_progress, step_size, precision, max_steps)

1526 **try**:

-> 1527 inv_h_dot_g_T = spsolve(-h, g, assume_a="pos", check_finite=False)

1528 **except** (ValueError, LinAlgError) **as** e:

File /opt/anaconda3/envs/cpsc330/lib/python3.10/site-packages/scipy/linalg/_basic.py:254, in solve(a, b, sym_pos, lower, overwrite_a, overwrite_b, check_finite, assume_a, transposed)

251 lu, x, info = posv(a1, b1, lower=lower,

252 overwrite_a=overwrite_a,

253 overwrite_b=overwrite_b)

- Ok, going that [that URL](https://lifelines.readthedocs.io/en/latest/Examples.html#problems-with-convergence-in-the-cox-proportional-hazard-model) (<https://lifelines.readthedocs.io/en/latest/Examples.html#problems-with-convergence-in-the-cox-proportional-hazard-model>), it seems the easiest solution is to add a penalizer.
 - FYI this is related to switching from `LinearRegression` to `Ridge`.
 - Adding `drop='first'` on our OHE might have helped with this.
 - (For 340 folks: we're adding regularization; `lifelines` adds both L1 and L2 regularization, aka elastic net)

In [73]:

```

1 cph = lifelines.CoxPHFitter(penalizer=0.1)
2 cph.fit(train_df_surv, duration_col="tenure", event_col="Churn");
```

We can look at the coefficients learned by the model and start interpreting them!

```
In [74]: 1 cph_params = pd.DataFrame(cph.params_).sort_values(by="coef", ascending
2 cph_params
```

Out[74]:

covariate	coef
Contract_Month-to-month	0.812875
OnlineSecurity_No	0.311151
OnlineBackup_No	0.298561
PaymentMethod_Electronic check	0.280801
Partner_No	0.244814
...	...
OnlineBackup_Yes	-0.282600
PaymentMethod_Credit card (automatic)	-0.302801
OnlineSecurity_Yes	-0.330346
Contract_One year	-0.351821
Contract_Two year	-0.776427

43 rows × 1 columns

- Looks like month-to-month leads to more churn, two-year contract leads to less churn; this makes sense!!!

```
In [80]: 1 # cph.baseline_hazard_ # baseline hazard
```

```
In [79]: 1 # cph.summary
```

Could we have gotten this type of information out of sklearn?

```
In [77]: 1 y_train.head()
```

Out[77]:

6464	No
5707	No
3442	No
3932	Yes
6124	No

Name: Churn, dtype: object

In [78]: 1 X_train.drop(columns=["tenure"]).head()

Out[78]:

	customerID	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	Internet
6464	4726-DLWQN	Male	1	No	No	Yes		Yes
5707	4537-DKTAL	Female	0	No	No	Yes		No
3442	0468-YRPXN	Male	0	No	No	Yes	No	Fit
3932	1304-NECVQ	Female	1	No	No	Yes	Yes	Fit
6124	7153-CHRBV	Female	0	Yes	Yes	Yes		No

I'm redefining feature types and our preprocessor for our sanity.

In [81]: 1 numeric_features = ["MonthlyCharges", "TotalCharges"]
2 drop_features = ["customerID", "tenure"]
3 passthrough_features = ["SeniorCitizen"]
4 target_column = ["Churn"]
5 # the rest are categorical
6 categorical_features = list(
7 set(train_df.columns)
8 - set(numeric_features)
9 - set(passthrough_features)
10 - set(drop_features)
11 - set(target_column)
12)

In [82]: 1 preprocessor = make_column_transformer(
2 (
3 make_pipeline(SimpleImputer(strategy="median"), StandardScaler(
4 numeric_features,
5)),
6 (OneHotEncoder(handle_unknown="ignore"), categorical_features),
7 ("passthrough", passthrough_features),
8 ("drop", drop_features),
9))

In [83]: 1 preprocessor.fit(X_train);

In [86]: 1 new_columns = (
2 numeric_features
3 + preprocessor.named_transformers_["onehotencoder"]
4 .get_feature_names_out(categorical_features)
5 .tolist()
6 + passthrough_features
7)

```
In [87]: 1 lr = make_pipeline(preprocessor, LogisticRegression(max_iter=1000))
2 lr.fit(X_train, y_train)
3 lr_coefs = pd.DataFrame(
4     data=np.squeeze(lr[1].coef_), index=new_columns, columns=["Coefficient"]
5 )
```

```
In [88]: 1 lr_coefs.sort_values(by="Coefficient", ascending=False)
```

Out[88]:

	Coefficient
Contract_Month-to-month	0.787653
InternetService_Fiber optic	0.600509
OnlineSecurity_No	0.291008
StreamingTV_Yes	0.258659
PaymentMethod_Electronic check	0.251646
...	...
MultipleLines_No	-0.169654
PaymentMethod_Credit card (automatic)	-0.204406
InternetService_DSL	-0.461593
TotalCharges	-0.743315
Contract_Two year	-0.765519

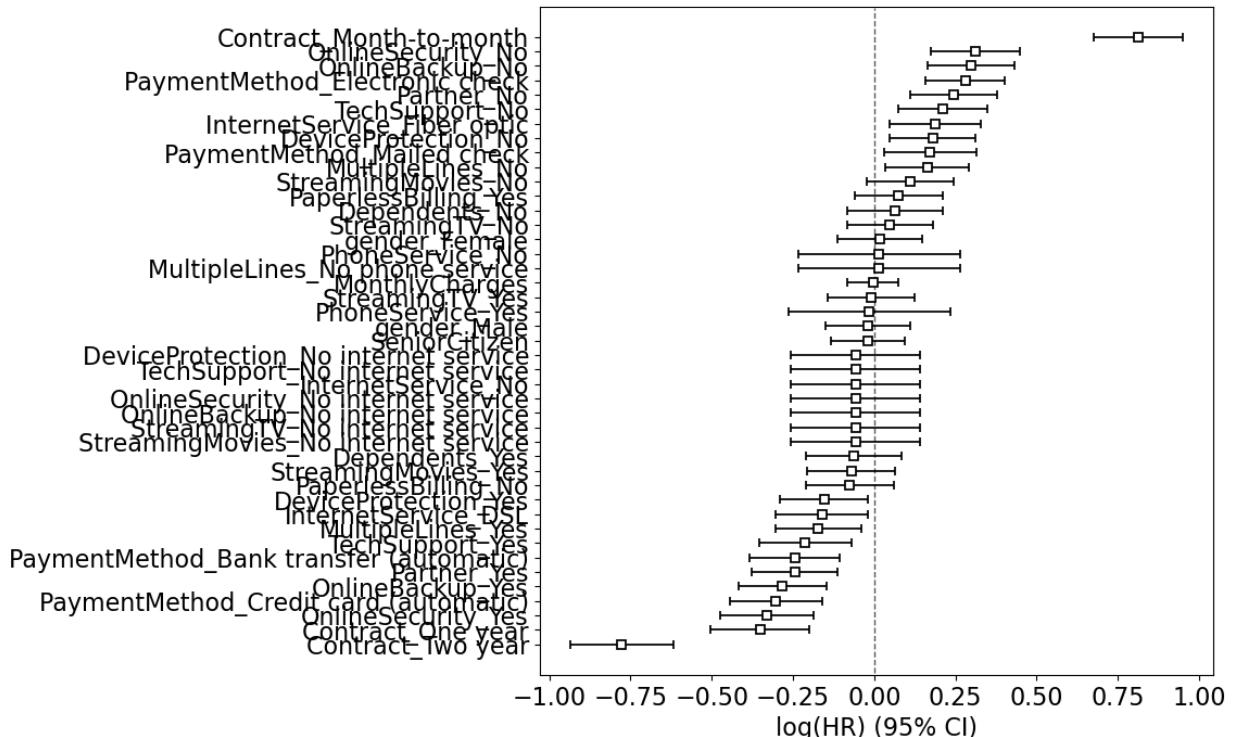
44 rows × 1 columns

- There is some agreement, which is good.
- But our survival model is much more useful.
 - Not to mention more correct.

- One thing we get with `lifelines` is confidence intervals on the coefficients:

In [89]:

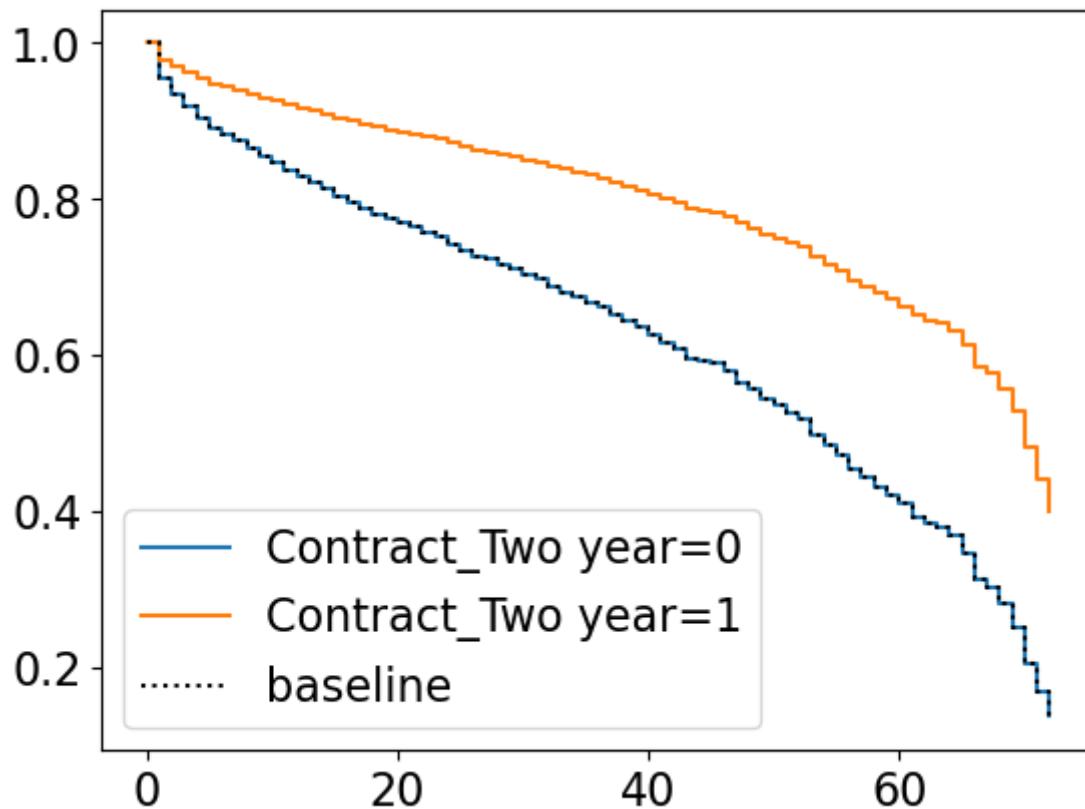
```
1 plt.figure(figsize=(8, 8))
2 cph.plot();
```



- (We could probably get the same for logistic regression if using `statsmodels` instead of `sklearn`.)
- However, in general, I would be careful with all of this.
- Ideally we would have more statistical training when using `lifelines` - there is a lot that can go wrong.
 - It comes with various diagnostics as well.
- But I think it's very useful to know about survival analysis and the availability of software to deal with it.
- Oh, and there are lots of other nice plots.

- Let's look at the survival plots for the people with
 - two-year contract (`Contract_Two year = 1`) and
 - people without two-year contract (`Contract_Two year = 0`)
- As expected, the former survive longer.

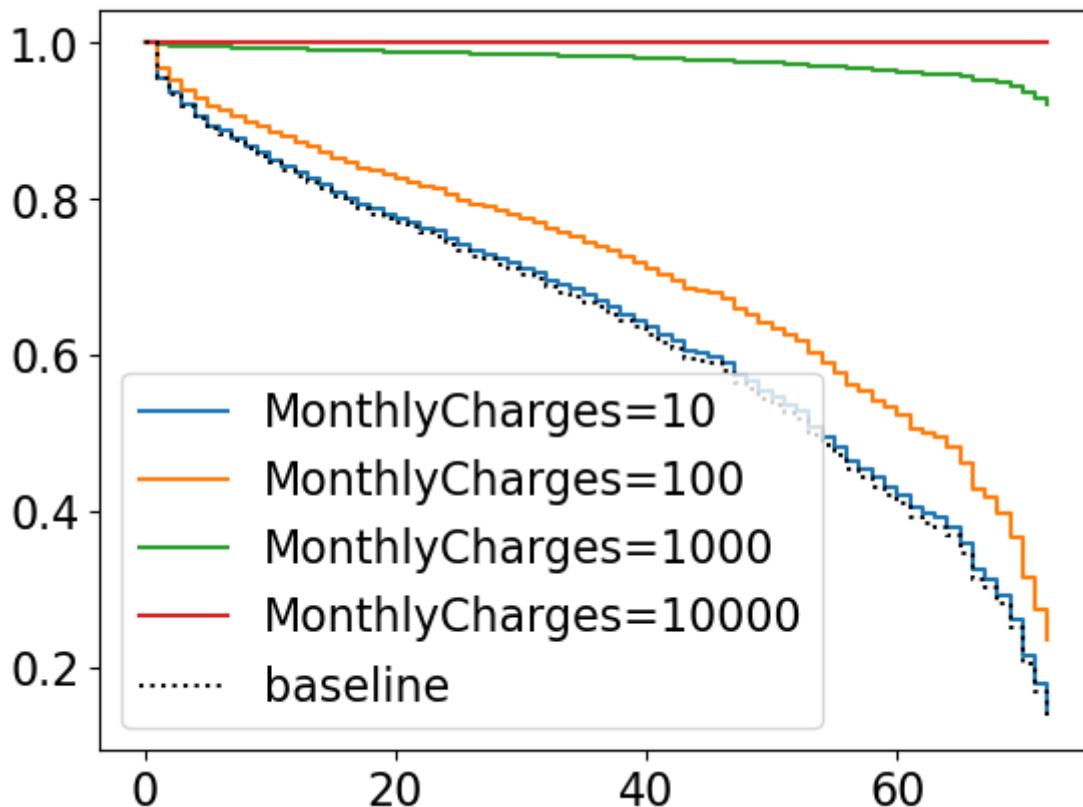
```
In [90]: 1 cph.plot_partial_effects_on_outcome("Contract_Two year", [0, 1]);
```



Now let's look at the survival plots for the people with different MonthlyCharges.

In [91]:

```
1 cph.plot_partial_effects_on_outcome("MonthlyCharges", [10, 100, 1000, 1]
```



- That's the thing with linear models, they can't stop the growth.
- We have a negative coefficient associated with `MonthlyCharges`

In [92]:

```
1 cph_params.loc["MonthlyCharges"]
```

Out[92]:

```
coef    -0.003185
Name: MonthlyCharges, dtype: float64
```

If your monthly charges are huge, it takes this to the extreme and thinks you'll basically never churn.

Prediction

- We can use survival analysis to make predictions as well.
- Here is the expected number of months to churn for the first 5 customers in the test set:

In [93]: 1 test_df_surv.drop(columns=["tenure", "Churn"]).head()

Out[93]:

	SeniorCitizen	MonthlyCharges	TechSupport_No	TechSupport_No internet service	TechSupport_Yes	MultipleLir
941	0.0	-1.154900	1.0	0.0	0.0	
1404	0.0	-1.383246	0.0	1.0	0.0	
5515	0.0	-1.514920	0.0	1.0	0.0	
3684	0.0	0.351852	1.0	0.0	0.0	
7017	0.0	-1.471584	0.0	1.0	0.0	

5 rows × 43 columns

In [94]: 1 test_df_surv.head()

Out[94]:

	Churn	tenure	SeniorCitizen	MonthlyCharges	TechSupport_No	TechSupport_No internet service	TechSupport_
941	0.0	13.0	0.0	-1.154900	1.0	0.0	
1404	0.0	35.0	0.0	-1.383246	0.0	1.0	
5515	0.0	18.0	0.0	-1.514920	0.0	1.0	
3684	0.0	43.0	0.0	0.351852	1.0	0.0	
7017	0.0	51.0	0.0	-1.471584	0.0	1.0	

5 rows × 45 columns

How long each non-churned customer is likely to stay according to the model assuming that they just joined right now?

In [95]: 1 cph.predict_expectation(test_df_surv).head() # assumes they just joined

Out[95]:

941	35.206724
1404	69.023086
5515	68.608565
3684	27.565062
7017	67.890933

dtype: float64

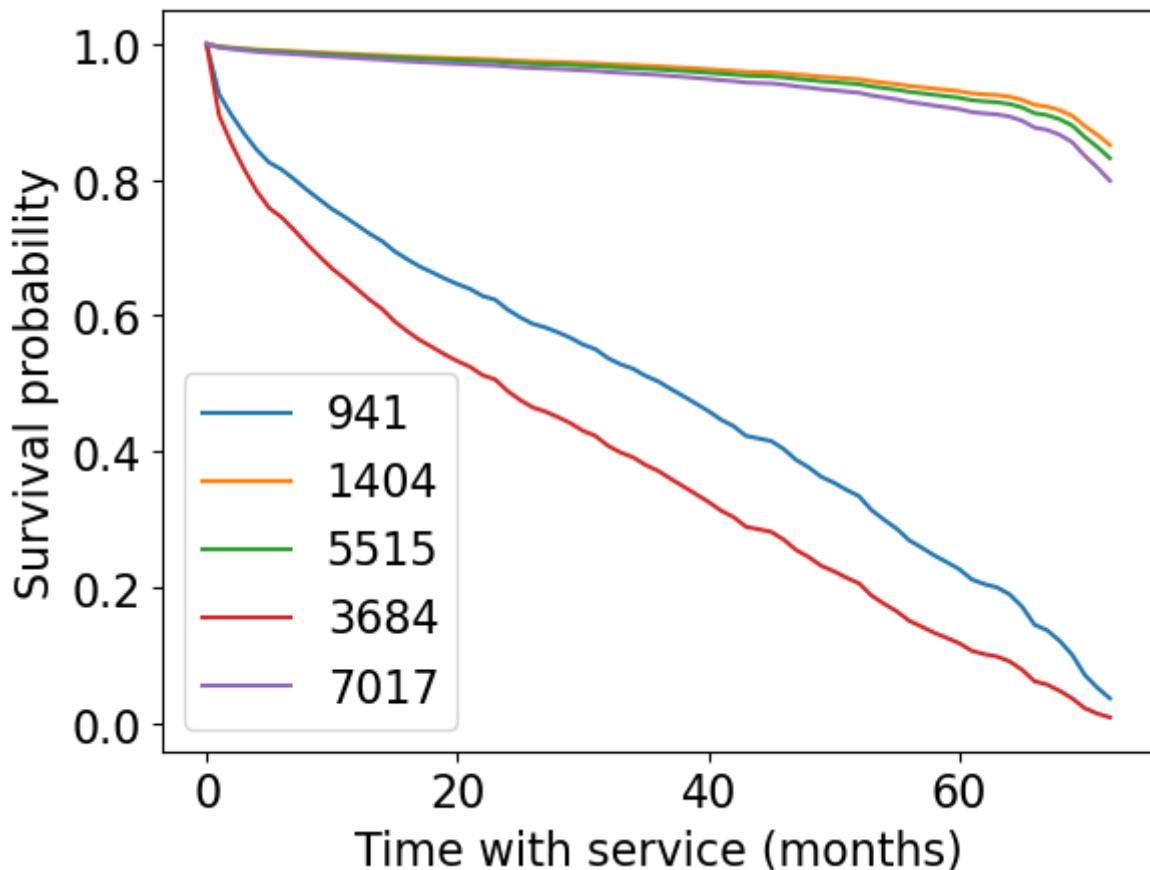
Survival curves for first 5 customers in the test set:

In [96]:

```

1 cph.predict_survival_function(test_df_surv[:5]).plot()
2 plt.xlabel("Time with service (months)")
3 plt.ylabel("Survival probability");

```



From `predict_survival_function` documentation:

Predict the survival function for individuals, given their covariates. This assumes that the individual just entered the study (that is, we do not condition on how long they have already lived for.)

So these curves are "starting now".

- There's no probability prerequisite for this course, so this is optional material.
- But you can do some interesting stuff here with conditional probabilities.
- "Given that a customer has been here 5 months, what's the outlook?"
 - It will be different than for a new customer.
 - Thus, we might still want to predict for the non-churned customers in the training set!
 - Not something we really thought about with our traditional supervised learning.

Let's get the customers who have not churned yet.

```
In [97]: 1 train_df_surv_not_churned = train_df_surv[train_df_surv["Churn"] == 0]
```

We can condition on the person having been around for 20 months.

```
In [98]: 1 cph.predict_survival_function(train_df_surv_not_churned[:1], conditiona
```

Out[98]:

6464

0.0 1.000000

1.0 0.996788

2.0 0.991966

3.0 0.989443

4.0 0.982570

... ...

68.0 0.429634

69.0 0.429634

70.0 0.429634

71.0 0.429634

72.0 0.429634

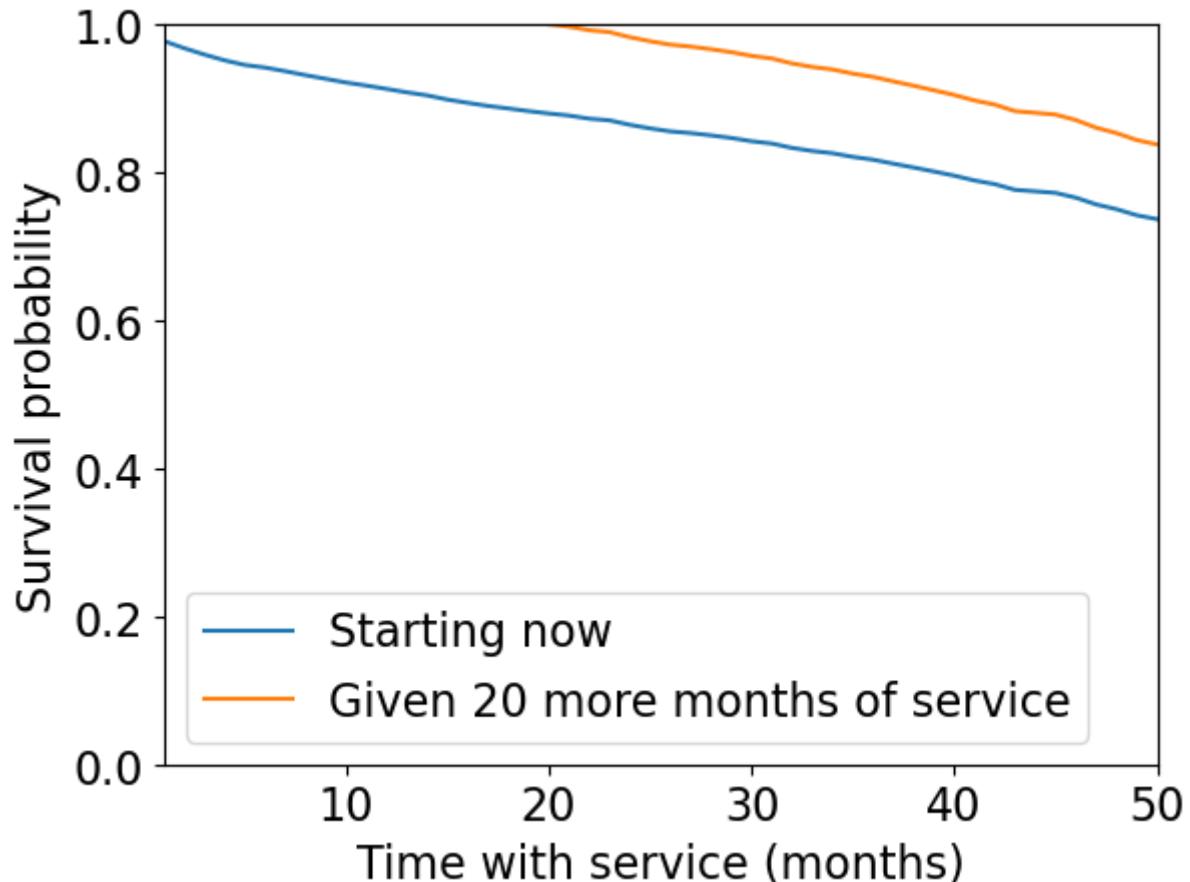
73 rows × 1 columns

In [99]:

```

1 plt.figure()
2 cph.predict_survival_function(train_df_surv_not_churned[:1]).plot(ax=plt
3 preds = cph.predict_survival_function(
4     train_df_surv_not_churned[:1], conditional_after=20
5 )
6 plt.plot(preds.index[20:], preds.values[:-20])
7 plt.xlabel("Time with service (months)")
8 plt.ylabel("Survival probability")
9 plt.legend(["Starting now", "Given 20 more months of service"])
10 plt.ylim([0, 1])
11 plt.xlim([1, 50]);

```



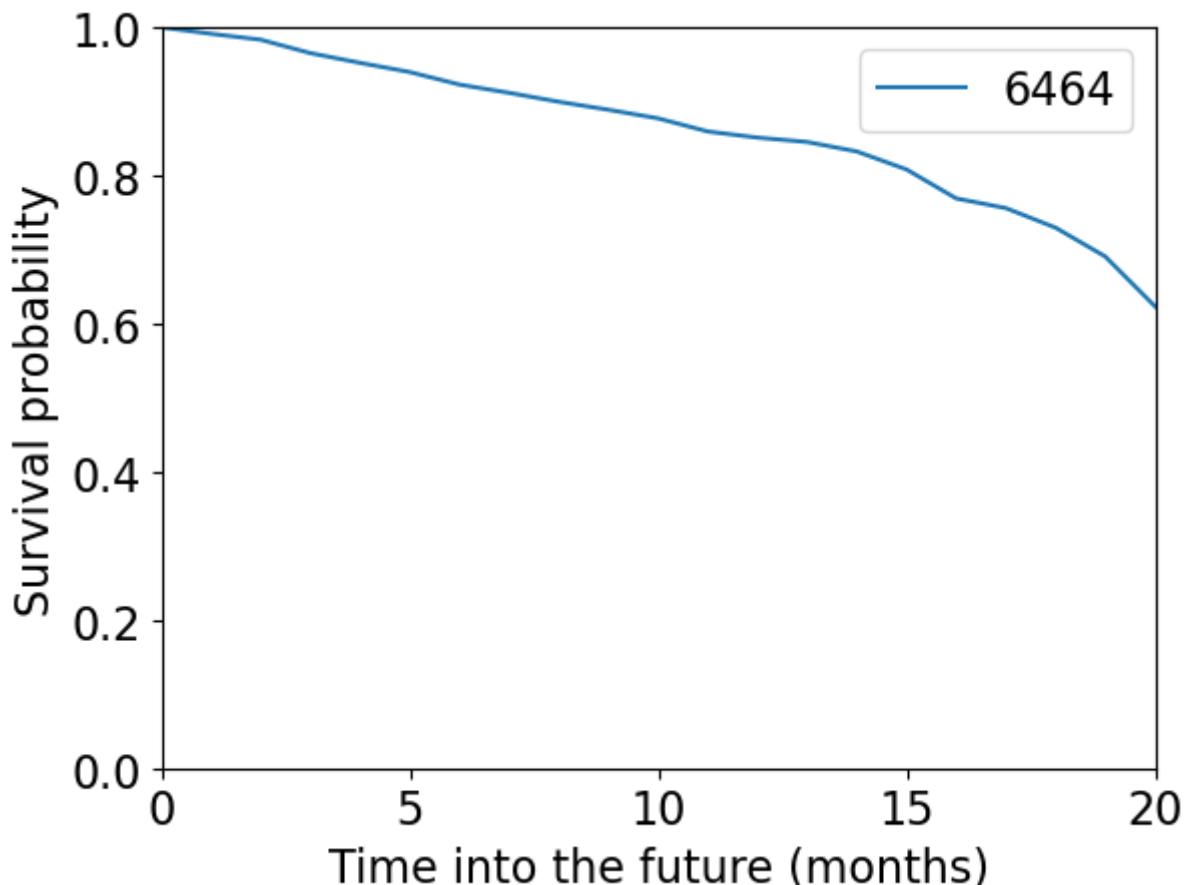
- Look at how the survival function (and expected lifetime) is much longer *given* that the customer has already lasted 20 months.
- How long each non-churned customer is likely to stay according to the model assuming that they have been here for the tenure time?
- So, we can set this to their actual tenure so far to get a prediction of what will happen going forward:

In [100]:

```

1 cph.predict_survival_function(
2     train_df_surv_not_churned[:1],
3     conditional_after=train_df_surv_not_churned[:1][ "tenure" ],
4 ).plot()
5 plt.xlabel("Time into the future (months)")
6 plt.ylabel("Survival probability")
7 plt.ylim([0, 1])
8 plt.xlim([0, 20]);

```



- Another useful application: you could ask what is the [customer lifetime value](https://en.wikipedia.org/wiki/Customer_lifetime_value) (https://en.wikipedia.org/wiki/Customer_lifetime_value).
 - Basically, how much money do you expect to make off this customer between now and when they churn?
- With regular supervised learning, tenure was a feature and we could only predict whether or not they had churned by then.

In []:

1

Evaluation

By default score returns "partial log likelihood":

```
In [101]: 1 cph.score(train_df_surv)
```

Out[101]: -1.8641864337292489

```
In [102]: 1 cph.score(test_df_surv)
```

Out[102]: -1.7277854625841886

We can look at the "concordance index" which is more interpretable:

```
In [103]: 1 cph.concordance_index_
```

Out[103]: 0.8625888648969532

```
In [104]: 1 cph.score(train_df_surv, scoring_method="concordance_index")
```

Out[104]: 0.8625888648969532

```
In [105]: 1 cph.score(test_df_surv, scoring_method="concordance_index")
```

Out[105]: 0.8546143543902771

From the documentation [here](#)

(<https://lifelines.readthedocs.io/en/latest/Survival%20Regression.html#model-selection-and-calibration-in-survival-regression>):

Another censoring-sensitive measure is the concordance-index, also known as the c-index. This measure evaluates the accuracy of the ranking of predicted time. It is in fact a generalization of AUC, another common loss function, and is interpreted similarly:

- 0.5 is the expected result from random predictions,
- 1.0 is perfect concordance and,
- 0.0 is perfect anti-concordance (multiply predictions with -1 to get 1.0)

[Here](https://stats.stackexchange.com/a/478305/11867) (<https://stats.stackexchange.com/a/478305/11867>) is an excellent introduction & description of the c-index for new users.

```
In [96]: 1 # cph.log_likelihood_ratio_test()
```

```
In [97]: 1 # cph.check_assumptions(df_train_surv)
```

Other approaches / what did we not cover?

There are many other approaches to modelling in survival analysis:

- Time-varying proportional hazards.
 - What if some of the features change over time, e.g. plan type, number of lines, etc.
- Approaches based on deep learning, e.g. the [pysurvival](https://square.github.io/pysurvival/) (<https://square.github.io/pysurvival/>) package.
- Random survival forests.
- And more...

Types of censoring

There are also various types and sub-types of censoring we didn't cover:

- What we saw today is data with "right censoring"
- Sub-types within right censoring
 - Did everyone join at the same time?
 - Other there other reasons the data might be censored at random times, e.g. the person died?
- Left censoring
- Interval censoring

Summary

- Censoring and incorrect approaches to handling it
 - Throw away people who haven't churned
 - Assume everyone churns today
- Predicting tenure vs. churned
- Survival analysis encompasses both of these, and deals with censoring
- And it can make rich and interesting predictions!
- KM model -> doesn't look at features
- CPH model -> like linear regression, does look at the features

True/False questions

1. If all customers joined a service at the same time (hypothetically), then censoring would not be an issue.
2. The Cox proportional hazards model (`cph` above) assumes the effect of a feature is the same for all customers and over all time.
3. Survival analysis can be useful even without a "deployment" stage.

References

Some people working with this same dataset:

- <https://medium.com/@zachary.james.angell/applying-survival-analysis-to-customer-churn-40b5a809b05a> (<https://medium.com/@zachary.james.angell/applying-survival-analysis-to-customer-churn-40b5a809b05a>)
- <https://towardsdatascience.com/churn-prediction-and-prevention-in-python-2d454e5fd9a5> (<https://towardsdatascience.com/churn-prediction-and-prevention-in-python-2d454e5fd9a5>) (Cox)
- <https://towardsdatascience.com/survival-analysis-in-python-a-model-for-customer-churn-e737c5242822> (<https://towardsdatascience.com/survival-analysis-in-python-a-model-for-customer-churn-e737c5242822>)
- <https://towardsdatascience.com/survival-analysis-intuition-implementation-in-python-504fde4fcf8e> (<https://towardsdatascience.com/survival-analysis-intuition-implementation-in-python-504fde4fcf8e>)

lifelines documentation:

- <https://lifelines.readthedocs.io/en/latest/Survival%20analysis%20with%20lifelines.html> (<https://lifelines.readthedocs.io/en/latest/Survival%20analysis%20with%20lifelines.html>)
- <https://lifelines.readthedocs.io/en/latest/Survival%20Analysis%20intro.html#introduction-to-survival-analysis> (<https://lifelines.readthedocs.io/en/latest/Survival%20Analysis%20intro.html#introduction-to-survival-analysis>)

CPSC 330

Applied Machine Learning

Lecture 21: Ethics

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

In [29]:

```
1 import os
2 import sys
3
4 import IPython
5 import matplotlib.pyplot as plt
# import mglearn
6 import numpy as np
7 import pandas as pd
8 from IPython.display import HTML, display
9 from sklearn.dummy import DummyClassifier
10 from sklearn.linear_model import LogisticRegression
11 from sklearn.metrics import accuracy_score, f1_score, precision_score,
12 from sklearn.model_selection import cross_val_score, cross_validate, train_test_split
13 from sklearn.pipeline import Pipeline, make_pipeline
14 from sklearn.preprocessing import StandardScaler
15
16 %matplotlib inline
17 pd.set_option("display.max_colwidth", 200)
18
19 from IPython.display import Image
```

Lecture plan

- Guest lecture by Dr. Giulia Toti!
- ML fairness activity

ML fairness activity

AI/ML systems can give the illusion of objectivity as they are derived from seemingly unbiased data & algorithm. However, humans are inherently biased and AI/ML systems, if not carefully evaluated, can even further amplify the existing inequities and systemic bias in our society.

How do we make sure our AI/ML systems are *fair*? Which metrics can we use to quantify 'fairness' in AI/ML systems?

Let's examine this on [the adult census data set \(<https://www.kaggle.com/uciml/adult-census-income>\)](https://www.kaggle.com/uciml/adult-census-income).

```
In [30]: 1 census_df = pd.read_csv("../data/adult.csv")
          2 census_df.shape
```

Out[30]: (32561, 15)

```
In [31]: 1 train_df, test_df = train_test_split(census_df, test_size=0.4, random_s
```

```
In [32]: 1 train_df
```

Out[32]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	
25823	36	Private	245521	7th-8th		4	Married-civ-spouse	Farming-fishing	Husband
10274	26	Private	134287	Assoc-voc		11	Never-married	Sales	Own-child
27652	25	Local-gov	109526	HS-grad		9	Married-civ-spouse	Craft-repair	Husband
13941	23	Private	131275	HS-grad		9	Never-married	Craft-repair	Own-child
31384	27	Private	193122	HS-grad		9	Married-civ-spouse	Machine-op-inspct	Husband
...
29802	25	Private	410240	HS-grad		9	Never-married	Craft-repair	Own-child
5390	51	Private	146767	Assoc-voc		11	Married-civ-spouse	Prof-specialty	Husband
860	55	Federal-gov	238192	HS-grad		9	Married-civ-spouse	Tech-support	Husband
15795	41	Private	154076	Some-college		10	Married-civ-spouse	Adm-clerical	Husband
23654	22	Private	162667	HS-grad		9	Never-married	Handlers-cleaners	Own-child

25823	36	Private	245521	7th-8th		4	Married-civ-spouse	Farming-fishing	Husband
10274	26	Private	134287	Assoc-voc		11	Never-married	Sales	Own-child
27652	25	Local-gov	109526	HS-grad		9	Married-civ-spouse	Craft-repair	Husband
13941	23	Private	131275	HS-grad		9	Never-married	Craft-repair	Own-child
31384	27	Private	193122	HS-grad		9	Married-civ-spouse	Machine-op-inspct	Husband
...
29802	25	Private	410240	HS-grad		9	Never-married	Craft-repair	Own-child
5390	51	Private	146767	Assoc-voc		11	Married-civ-spouse	Prof-specialty	Husband
860	55	Federal-gov	238192	HS-grad		9	Married-civ-spouse	Tech-support	Husband
15795	41	Private	154076	Some-college		10	Married-civ-spouse	Adm-clerical	Husband
23654	22	Private	162667	HS-grad		9	Never-married	Handlers-cleaners	Own-child

19536 rows × 15 columns

```
In [33]: 1 train_df_nan = train_df.replace("?", np.nan)
          2 test_df_nan = test_df.replace("?", np.nan)
          3 train_df_nan.shape
```

Out[33]: (19536, 15)

```
In [34]: 1 # Let's identify numeric and categorical features
          2
          3 numeric_features = [
          4     "age",
          5     "fnlwgt",
          6     "capital.gain",
          7     "capital.loss",
          8     "hours.per.week",
          9 ]
          10
          11 categorical_features = [
          12     "workclass",
          13     "marital.status",
          14     "occupation",
          15     "relationship",
          16     "# race",
          17     "native.country",
          18 ]
          19
          20 ordinal_features = ["education"]
          21 binary_features = [
          22     "sex"
          23 ] # Not binary in general but in this particular dataset it seems to have
          24 drop_features = ["education.num"]
          25 target = "income"
```

```
In [35]: 1 train_df["education"].unique()
```

Out[35]: array(['7th-8th', 'Assoc-voc', 'HS-grad', 'Bachelors', 'Some-college',
 '10th', '11th', 'Prof-school', '12th', '5th-6th', 'Masters',
 'Assoc-acdm', '9th', 'Doctorate', '1st-4th', 'Preschool'],
 dtype=object)

In [36]:

```

1 education_levels = [
2     "Preschool",
3     "1st-4th",
4     "5th-6th",
5     "7th-8th",
6     "9th",
7     "10th",
8     "11th",
9     "12th",
10    "HS-grad",
11    "Prof-school",
12    "Assoc-voc",
13    "Assoc-acdm",
14    "Some-college",
15    "Bachelors",
16    "Masters",
17    "Doctorate",
18 ]

```

In [37]:

```
1 assert set(education_levels) == set(train_df["education"].unique())
```

In [38]:

```

1 X_train = train_df_nan.drop(columns=[target])
2 y_train = train_df_nan[target]
3
4 X_test = test_df_nan.drop(columns=[target])
5 y_test = test_df_nan[target]

```

In [39]:

```

1 from sklearn.compose import ColumnTransformer, make_column_transformer
2 from sklearn.impute import SimpleImputer
3 from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
4
5 numeric_transformer = make_pipeline(StandardScaler())
6
7 ordinal_transformer = OrdinalEncoder(categories=[education_levels], dtype='int')
8
9 categorical_transformer = make_pipeline(
10     SimpleImputer(strategy="constant", fill_value="missing"),
11     OneHotEncoder(handle_unknown="ignore", sparse=False),
12 )
13
14 binary_transformer = make_pipeline(
15     SimpleImputer(strategy="constant", fill_value="missing"),
16     OneHotEncoder(drop="if_binary", dtype=int),
17 )
18
19 preprocessor = make_column_transformer(
20     (numeric_transformer, numeric_features),
21     (ordinal_transformer, ordinal_features),
22     (binary_transformer, binary_features),
23     (categorical_transformer, categorical_features),
24     ("drop", drop_features),
25 )

```

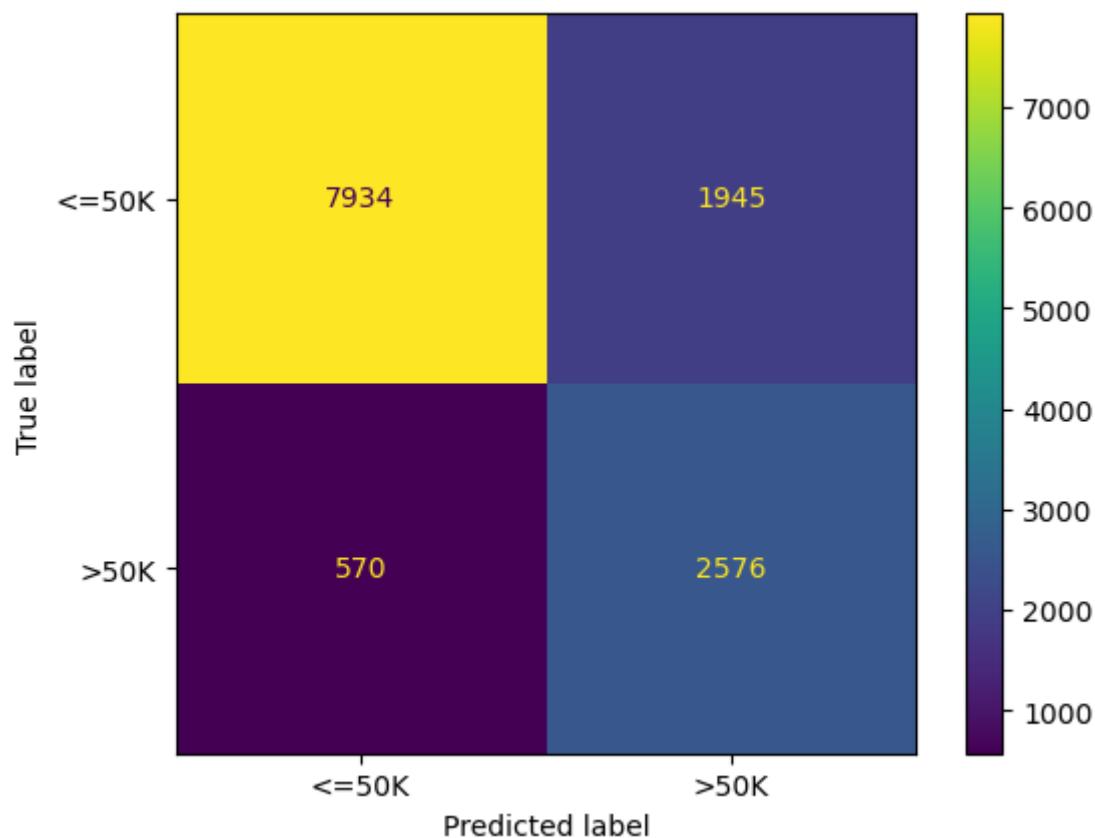
```
In [40]: 1 y_train.value_counts()
```

```
Out[40]: <=50K    14841
>50K      4695
Name: income, dtype: int64
```

```
In [41]: 1 pipe_lr = make_pipeline(
2     preprocessor, LogisticRegression(class_weight="balanced", max_iter=
3 )
```

```
In [42]: 1 pipe_lr.fit(X_train, y_train);
```

```
In [43]: 1 from sklearn.metrics import ConfusionMatrixDisplay # Recommended method
2
3 ConfusionMatrixDisplay.from_estimator(pipe_lr, X_test, y_test);
```



Let's examine confusion matrix separately for the two genders we have in the data.

```
In [44]: 1 X_train_enc = preprocessor.fit_transform(X_train)
2 preprocessor.named_transformers_["pipeline-2"]["onehotencoder"].get_fea
```

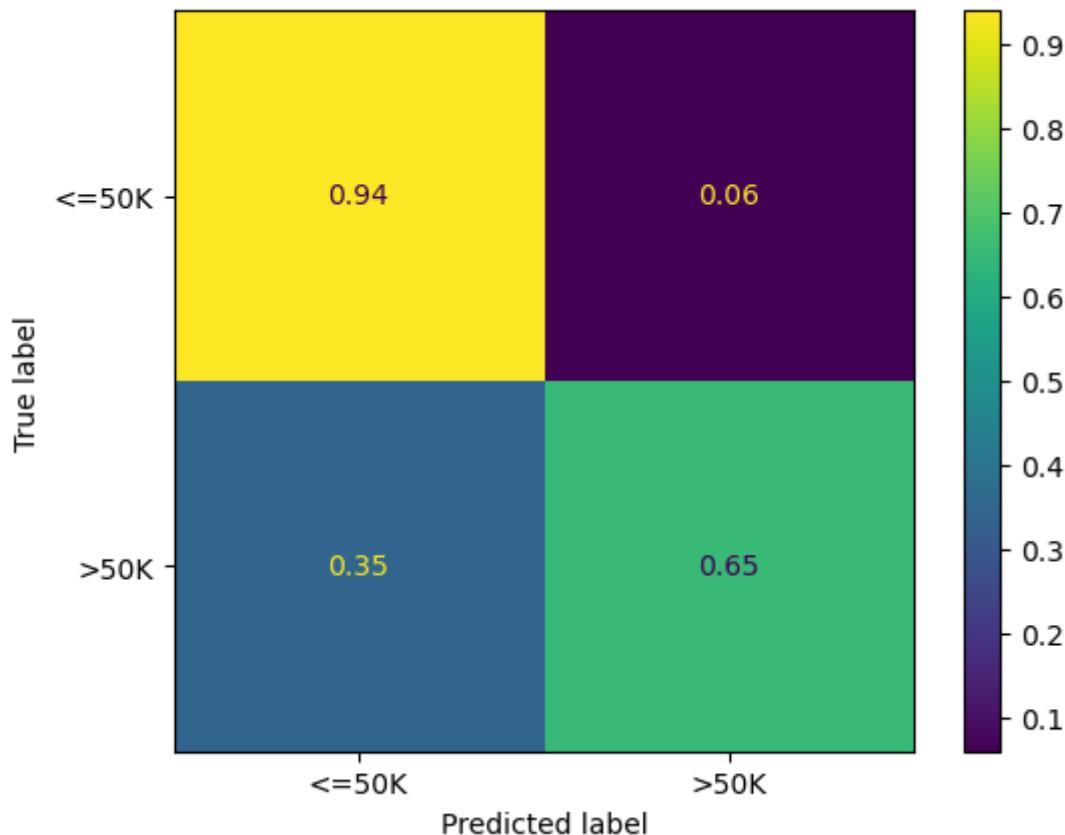
```
Out[44]: array(['x0_Male'], dtype=object)
```

In [45]: 1 X_test.head()

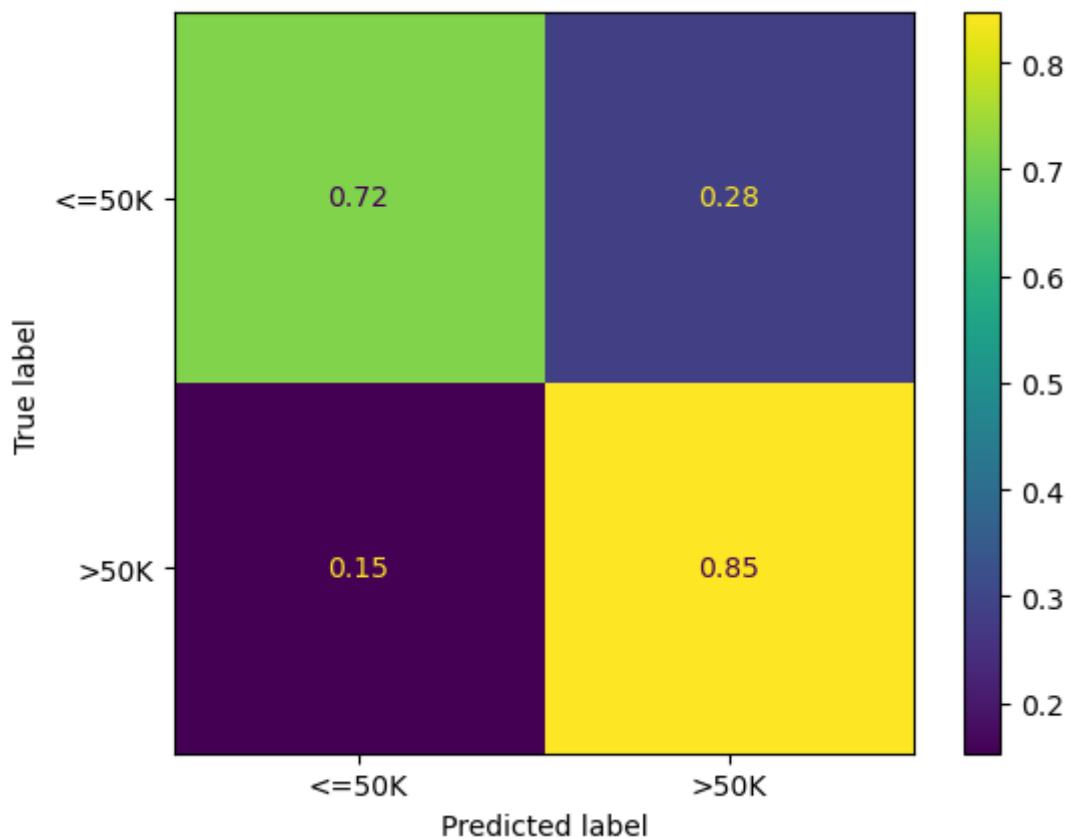
	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship
14160	29	Private	280618	Some-college	10	Married-civ-spouse	Handlers-cleaners	Husband \
27048	19	Private	439779	Some-college	10	Never-married	Sales	Own-child \
28868	28	Private	204734	Some-college	10	Married-civ-spouse	Tech-support	Wife \
5667	35	Private	107991	11th	7	Never-married	Sales	Not-in-family \
7827	20	Private	54152	Some-college	10	Never-married	Adm-clerical	Own-child \

In [46]: 1 X_female = X_test.query("sex=='Female'")
2 X_male = X_test.query("sex=='Male'")
3
4 y_female = y_test[X_female.index]
5 y_male = y_test[X_male.index]
6 female_preds = pipe_lr.predict(X_female)
7 male_preds = pipe_lr.predict(X_male)

In [47]: 1 ConfusionMatrixDisplay.from_estimator(pipe_lr, X_female, y_female, norm



```
In [48]: 1 ConfusionMatrixDisplay.from_estimator(pipe_lr, X_male, y_male, normalize=True)
```



What's the accuracy of this model?

```
In [49]: 1 from sklearn.metrics import confusion_matrix
2
3 data = {"male": [], "female": []}
4 f_TN, f_FP, f_FN, f_TP = confusion_matrix(y_female, female_preds).ravel()
5 m_TN, m_FP, m_FN, m_TP = confusion_matrix(y_male, male_preds).ravel()
```

```
In [50]: 1 accuracy_male = accuracy_score(y_male, male_preds)
2 accuracy_female = accuracy_score(y_female, female_preds)
3 data["male"].append(accuracy_male)
4 data["female"].append(accuracy_female)
5
6 print("Accuracy male: {:.3f}".format(accuracy_male))
7 print("Accuracy female: {:.3f}".format(accuracy_female))
```

Accuracy male: 0.756
 Accuracy female: 0.909

```
In [ ]:
```

```
In [51]: 1 y_female.value_counts(normalize=True)
```

```
Out[51]: <=50K    0.892675
>50K     0.107325
Name: income, dtype: float64
```

```
In [52]: 1 y_male.value_counts(normalize=True)
```

```
Out[52]: <=50K    0.691999
>50K     0.308001
Name: income, dtype: float64
```

There is more class imbalance for female!

Let's assume that a company is using this classifier for loan approval with a simple rule that if the income is $\geq 50K$, approve the loan else reject the loan.

Statistical parity suggests that the proportion of each segment of a protected class (e.g. sex) should receive the positive outcome at equal rates. For example, the number of loans approved for female should be equal to male.

Calculate the precision for male and female. Based on your results, do you think this income classifier is fair?

```
In [53]: 1 precision_male = precision_score(y_male, male_preds, pos_label=>">50K")
2 precision_female = precision_score(y_female, female_preds, pos_label=>>50K)
3 data["male"].append(precision_male)
4 data["female"].append(precision_female)
5
6 print("Precision male: {:.3f}".format(precision_male))
7 print("Precision female: {:.3f}".format(precision_female))
```

```
Precision male: 0.570
Precision female: 0.567
```

Equal opportunity suggests that each group should get the positive outcome at equal rates, assuming that people in this group qualify for it. For example, if a man and a woman have both a certain level of income, we want them to have the same chance of getting the loan. In other words, the true positive rate (TPR or recall) of both groups should be equal.

```
In [54]: 1 recall_male = recall_score(y_male, male_preds, pos_label=>">50K")
2 recall_female = recall_score(y_female, female_preds, pos_label=>>50K)
3
4 data["male"].append(recall_male)
5 data["female"].append(recall_female)
6
7 print("Recall male: {:.3f}".format(recall_male))
8 print("Recall female: {:.3f}".format(recall_female))
```

```
Recall male: 0.847
Recall female: 0.654
```

There is usually a tradeoff between rationality (adopting effective means to achieve your desired outcome) and bias. The desired outcome of banks, for example, is to maximize their profit. So in many circumstances, they not only care about approving as many qualified applications as possible (true positive), but also to avoid approving unqualified applications (false positive) because default loan could have detrimental effects for them.

Let's examine false positive rate (FPR) of both groups.

```
In [55]: 1 fpr_male = m_FP / (m_FP + m_TN)
2 fpr_female = f_FP / (f_FP + f_TN)
3
4 data["male"].append(fpr_male)
5 data["female"].append(fpr_female)
6
7 print("FPR male: {:.3f}".format(fpr_male))
8 print("FPR female: {:.3f}".format(fpr_female))
```

FPR male: 0.284
 FPR female: 0.060

```
In [56]: 1 pd.DataFrame(data, index=["accuracy", "precision", "recall", "FPR"])
```

	male	female
accuracy	0.756170	0.909365
precision	0.570103	0.567416
recall	0.847186	0.654428
FPR	0.284340	0.059984

- Discuss these results with your neighbours.
- Does the effect still exist if the sex feature is removed from the model (but you still have it available separately to do the two confusion matrices)?

```
In [ ]: 1
```

```
In [ ]: 1
```

Lecture 21: Communication

UBC 2022-23

Instructor: Mathias Lécuyer

Imports

In [2]:

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import pandas as pd
4 from sklearn.compose import ColumnTransformer, TransformedTargetRegressor
5 from sklearn.dummy import DummyClassifier, DummyRegressor
6 from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
7 from sklearn.impute import SimpleImputer
8 from sklearn.linear_model import Ridge
9 from sklearn.metrics import log_loss
10 from sklearn.model_selection import (
11     GridSearchCV,
12     cross_val_score,
13     cross_validate,
14     train_test_split,
15 )
16 from sklearn.pipeline import Pipeline, make_pipeline
17 from sklearn.preprocessing import (
18     MinMaxScaler,
19     OneHotEncoder,
20     OrdinalEncoder,
21     StandardScaler,
22 )
23
24 plt.rcParams["font.size"] = 16
```

Learning objectives

- When communicating about applied ML, tailor an explanation to the intended audience.
- Apply best practices of technical communication, such as bottom-up explanations and reader-centric writing.
- Given an ML problem, analyze the decision being made and the objectives.
- Avoid the pitfall of thinking about ML as coding in isolation; build the habit of relating your work to the surrounding context and stakeholders.
- Interpret a confidence score or credence, e.g. what does it mean to be 5% confident that a statement is true.
- Maintain a healthy skepticism of `predict_proba` scores and their possible interpretation as credences.
- Be careful and precise when communicating confidence to stakeholders in an ML project.

- Identify misleading visualizations.

Attribution

- The first part of this lecture is adapted from [DSCI 542 \(\[https://github.com/UBC-MDS/DSCI_542_comm-arg\]\(https://github.com/UBC-MDS/DSCI_542_comm-arg\)\)](https://github.com/UBC-MDS/DSCI_542_comm-arg), created by [David Laing \(<https://davidklaing.com/>\)](https://davidklaing.com/).
- The visualization component of this lecture benefitted from discussions with [Firas Moosvi \(<http://firas.moosvi.com/>\)](http://firas.moosvi.com/) about his course, [DSCI 531 \(\[https://github.com/UBC-MDS/DSCI_531_viz-1\]\(https://github.com/UBC-MDS/DSCI_531_viz-1\)\)](https://github.com/UBC-MDS/DSCI_531_viz-1).

Why should we care about effective communication?

- Most ML practitioners work in an organization with >1 people.
- There will very likely be stakeholders other than yourself.
- Those people need to understand what you're doing because:
 - their state of mind may change the way you do things (see below)
 - your state of mind may change the way they do things (interpreting your results)

ML suffers from some particular communication issues:

- overstating one's results / unable to articulate the limitations
- unable to explain the predictions
- Can we trust test error?
- Why did CatBoost make that prediction?
- What does it mean if `predict_proba` outputs 0.9?

These issues are there because these things are actually very hard to explain!

Activity: explaining GridSearchCV

Below are two possible explanations of `GridSearchCV` pitched to different audiences. Read them both and then follow the instructions at the end.

Explanation 1

Machine learning algorithms, like an airplane's cockpit, typically involve a bunch of knobs and switches that need to be set.



For example, check out the documentation of the popular random forest algorithm [here](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) (<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>). Here's a list of the function arguments, along with their default values (from the documentation):

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, criterion='gini',
max_depth=None, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True,
oob_score=False, n_jobs=None, random_state=None, verbose=0,
warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)
```

Holy cow, that's a lot of knobs and switches! As a machine learning practitioner, how am I supposed to choose `n_estimators`? Should I leave it at the default of 100? Or try 1000? What about `criterion` or `class_weight` for that matter? Should I trust the defaults?

Enter [GridSearchCV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) to save the day. The general strategy here is to choose the settings that perform best on the specific task of interest. So I can't say `n_estimators=100` is better than `n_estimators=1000` without knowing what problem I'm working on. For a specific problem, you usually have a numerical score that measures performance. `GridSearchCV` is part of the popular [scikit-learn](https://scikit-learn.org/) (<https://scikit-learn.org/>) Python machine learning library. It works by searching over various settings and tells you which one worked best on your problem.

The "grid" in "grid search" comes from the fact that it tries all possible combinations on a grid. For example, if you want it to consider setting `n_estimators` to 100, 150 or 200, and you want it to consider setting `criterion` to '`gini`' or '`entropy`', then it will search over all 6 possible combinations in a grid of 3 possible values by 2 possible values:

```
criterion='gini'    criterion='entropy'
```

In [3]:

```

1 # imports
2 from sklearn import datasets
3 from sklearn.ensemble import RandomForestClassifier
4 from sklearn.model_selection import GridSearchCV
5
6 # load a dataset
7 data = datasets.load_digits()
8 X = data["data"]
9 y = data["target"]
10
11 # set up the grid search
12 grid_search = GridSearchCV(
13     RandomForestClassifier(random_state=123),
14     param_grid={"n_estimators": [100, 150, 200], "criterion": ["gini",
15 ])
16
17 # run the grid search
18 grid_search.fit(X, y)
19 grid_search.best_params_

```

Out[3]: {'criterion': 'gini', 'n_estimators': 100}

As we can see from the output above, the grid search selected `criterion='gini'`, `n_estimators=100` , which was one of our 6 options above (specifically Option 1).

By the way, these "knobs" we've been setting are called [hyperparameters](#) ([https://en.wikipedia.org/wiki/Hyperparameter_\(machine_learning\)](https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning))) and the process of setting these hyperparameters automatically is called [hyperparameter optimization](#) (https://en.wikipedia.org/wiki/Hyperparameter_optimization) or [hyperparameter tuning](#).

~400 words, not including code.

Explanation 2

<https://medium.com/datadriveninvestor/an-introduction-to-grid-search-ff57adcc0998>
[\(https://medium.com/datadriveninvestor/an-introduction-to-grid-search-ff57adcc0998\)](https://medium.com/datadriveninvestor/an-introduction-to-grid-search-ff57adcc0998)

~400 words, not including code.

Discussion questions:

- What do you like about each explanation?
- What do you dislike about each explanation?
- What do you think is the intended audience for each explanation?
- Which explanation do you think is more effective overall for someone on Day 1 of CPSC 330?
- Each explanation has an image. Which one is more effective? What are the pros/cons?
- Each explanation has some sample code. Which one is more effective? What are the pros/cons?

After you're done reading, take ~5 min to consider the discussion questions above. Paste your answer to **at least one** of the above questions in the [Google jamboard](https://jamboard.google.com/d/1WbJTiNi-qt4EjyONcyO-CUxOhGxXhaRwYto28fC5Kzs/edit?usp=sharing) (<https://jamboard.google.com/d/1WbJTiNi-qt4EjyONcyO-CUxOhGxXhaRwYto28fC5Kzs/edit?usp=sharing>) under the appropriate question heading.

Principles of good explanations

Concepts *then* labels, not the other way around

The first explanation start with an analogy for the concept (and the label is left until the very end):

Machine learning algorithms, like an airplane's cockpit, typically involve a bunch of knobs and switches that need to be set.

In the second explanation, the first sentence is wasted on anyone who doesn't already know what "hyperparameter tuning" means:

Grid search is the process of performing hyper parameter tuning in order to determine the optimal values for a given model.

The effectiveness of these different statements depend on your audience.

See [this video](https://twitter.com/ProfFeynman/status/899963856549625858?s=20) (<https://twitter.com/ProfFeynman/status/899963856549625858?s=20>): "I learned very early the difference between knowing the name of something and knowing something." - Richard Feynman.

Bottom-up explanations

The [Curse of Knowledge](https://en.wikipedia.org/wiki/Curse_of_knowledge) (https://en.wikipedia.org/wiki/Curse_of_knowledge) leads to *top-down* explanations:

- When you know something well, you think about things in the context of all your knowledge.
- Those lacking the context, or frame of mind, cannot easily understand.

There is another way: *bottom-up* explanations:

When you're brand new to a concept, you benefit from analogies, concrete examples and familiar patterns.

New ideas in small chunks

The first explanation has a hidden conceptual skeleton:

1. The concept of setting a bunch of values.
2. Random forest example.
3. The problem / pain point.
4. The solution.
5. How it works - high level.
6. How it works - written example.
7. How it works - code example.
8. The name of what we were discussing all this time.

Reuse your running examples

Effective explanations often use the same example throughout the text and code. This helps readers follow the line of reasoning.

Approach from all angles

When we're trying to draw mental boundaries around a concept, it's helpful to see examples on all sides of those boundaries. If we were writing a longer explanation, it might have been better to show more, e.g.

- Performance with and without hyperparameter tuning.
- Other types of hyperparameter tuning (e.g. `RandomizedSearchCV`).

When experimenting, show the results asap

The first explanation shows the output of the code, whereas the second does not. This is easy to do and makes a big difference.

Interesting to you != useful to the reader (aka it's not about you)

Here is something which was deleted from the explanation:

Some hyperparameters, like `n_estimators` are numeric. Numeric hyperparameters are like the knobs in the cockpit: you can tune them continuously. `n_estimators` is numeric. Categorical hyperparameters are like the switches in the cockpit: they can take on (two or more) distinct values. `criterion` is categorical.

It's a very elegant analogy! But is it helpful?

And furthermore, what is my hidden motivation for wanting to include it? Elegance, art, and the pursuit of higher beauty? Or *making myself look smart*? So maybe another name for this principle could be **It's not about you**.

ML and decision-making

- There is often a wide gap between what people care about and what ML can do.
- To understand what ML can do, let's think about what **decisions** will be made using ML.

Decisions involve a few key pieces

- The **decision variable**: the variable that is manipulated through the decision.
 - E.g. how much should I sell my house for? (numeric)
 - E.g. should I sell my house? (categorical)
- The decision-maker's **objectives**: the variables that the decision-maker ultimately cares about, and wishes to manipulate indirectly through the decision variable.
 - E.g. my total profit, time to sale, etc.
- The **context**: the variables that mediate the relationship between the decision variable and the objectives.
 - E.g. the housing market, cost of marketing it, my timeline, etc.

How does this inform you as an ML practitioner?

Questions you have to answer:

- Who is the decision maker?
- What are their objectives?

- What are their alternatives?
- What is their context?
- What data do I need?

Break (10 min)

- We'll take a longer break today.
- Consider taking this time to fill out the instructor/TA evaluations if you haven't already.
Evaluation link(s):
 - [\(https://canvas.ubc.ca/courses/83420/external_tools/4732\)](https://canvas.ubc.ca/courses/83420/external_tools/4732)
 - [\(https://go.blueja.io/6smkkXqkVEq_u38wYKHE6Q\)](https://go.blueja.io/6smkkXqkVEq_u38wYKHE6Q).
- Here is [Mike's post on these evaluations](https://www.reddit.com/r/UBC/comments/k18qj7/teaching_evaluations_the_good_the_bad_and)
(https://www.reddit.com/r/UBC/comments/k18qj7/teaching_evaluations_the_good_the_bad_and)

Confidence and predict_proba

- What does it mean to be "confident" in your results?
- When you perform analysis, you are responsible for many judgment calls.
- [Your results will be different than others' \(https://fivethirtyeight.com/features/science-isnt-broken/#part1\)](https://fivethirtyeight.com/features/science-isnt-broken/#part1).
- As you make these judgments and start to form conclusions, how can you recognize your own uncertainties about the data so that you can communicate confidently?

What does this mean for us, when we're trying to make claims about our data?

Let's imagine that the following claim is true:

Vancouver has the highest cost of living of all cities in Canada.

Now let's consider a few beliefs we could hold:

1. Vancouver has the highest cost of living of all cities in Canada. **I am 95% sure of this.**
2. Vancouver has the highest cost of living of all cities in Canada. **I am 55% sure of this.**

The part in bold is called a [credence \(https://en.wikipedia.org/wiki/Credence_\(statistics\)\)](https://en.wikipedia.org/wiki/Credence_(statistics)). Which belief is better?

But what if it's actually Toronto that has the highest cost of living in Canada?

1. Vancouver has the highest cost of living of all cities in Canada. I am 95% sure of this.
2. Vancouver has the highest cost of living of all cities in Canada. I am 55% sure of this.

Which belief is better now?

Conclusion: We don't just want to be right. We want to be confident when we're right and hesitant when we're wrong.

What do credences mean in practical terms?

One of two things:

- **I would accept a bet at these odds.** 99% sure means, "For the chance of winning \$1, I would bet \$99 that I'm right about this." 75% sure means, "For the chance of winning \$25, I would bet \$75 that I'm right about this."
- **Long-run frequency of correctness.** 99% sure means, "For every 100 predictions I make at this level of confidence, I would expect only 1 of them to be incorrect." 75% sure means, "For every 100 predictions I make at this level of confidence, I would expect about 25 of them to be incorrect."

It's easy enough to evaluate how good we are at being right...

But if we want to evaluate *how good we are at knowing how right we are?*

We would need to keep track of not just the correctness of our predictions, but also our confidence in those predictions.

What does this have to do with applied ML?

- What if you predict that a credit card transaction is fraudulent?
 - We probably want predict_proba a lot of the time.
- What if predict_proba is 0.95 in that case?
 - How confident are YOU?
- What if you forecast that avocado prices will go up next week?
 - How confident are you there?
- Or what if you predict a house price to be \$800k?
 - That is not even a true/false statement.

Loss functions

When you call fit for LogisticRegression it has these same preferences: correct and confident > correct and hesitant > incorrect and hesitant > incorrect and confident.

```
In [3]: 1 from sklearn.metrics import log_loss
```

- This is a "loss" or "error" function like mean squared error, so lower values are better.
- When you call `fit` it tries to minimize this metric.

Correct and 95% confident:

```
In [4]: 1 log_loss(y_true=np.array([0]), y_pred=np.array([[0.95, 0.05]]), labels=
```

Out[4]: 0.05129329438755058

Correct and 55% confident:

```
In [5]: 1 log_loss(y_true=np.array([0]), y_pred=np.array([[0.55, 0.45]]), labels=
```

Out[5]: 0.5978370007556204

Incorrect and 55% confident:

```
In [6]: 1 log_loss(y_true=np.array([0]), y_pred=np.array([[0.45, 0.55]]), labels=
```

Out[6]: 0.7985076962177716

Incorrect and 95% confident:

```
In [7]: 1 log_loss(y_true=np.array([0]), y_pred=np.array([[0.05, 0.95]]), labels=
```

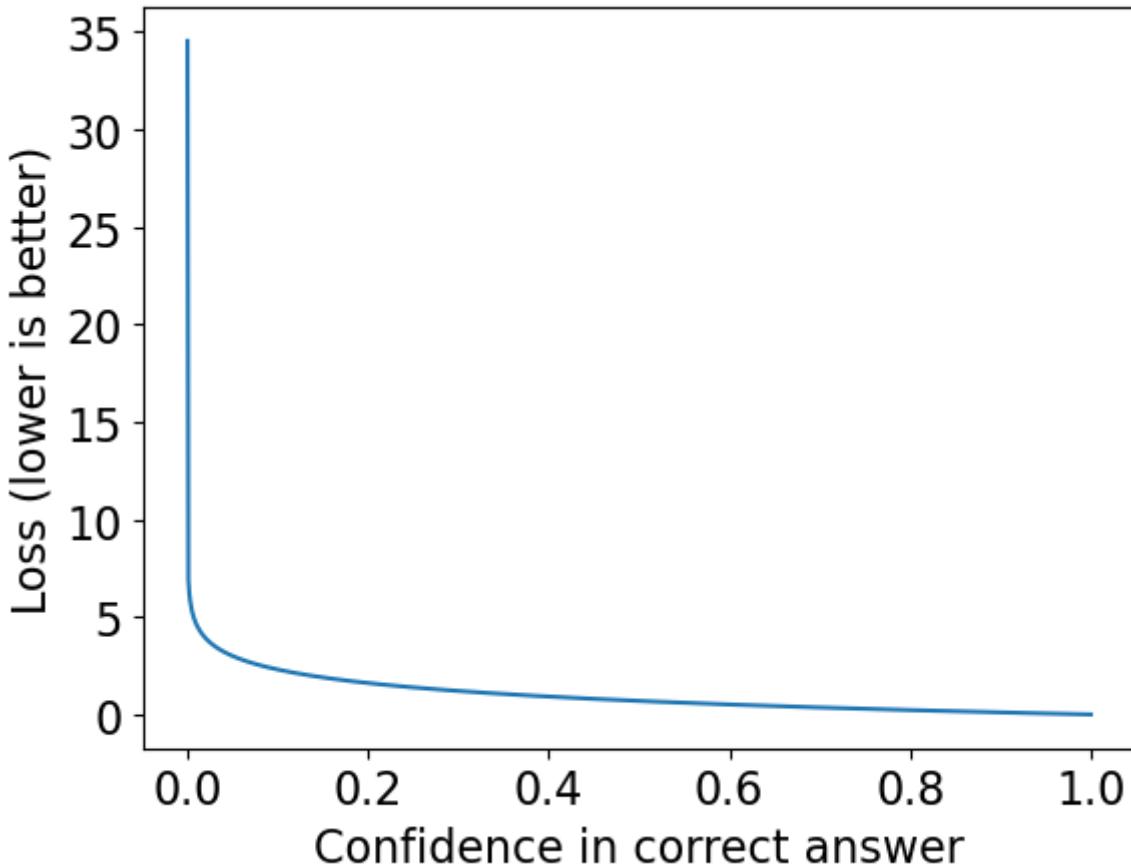
Out[7]: 2.995732273553991

In [8]:

```

1 grid = np.linspace(0, 1, 1000)
2 plt.plot(
3     grid,
4     [log_loss(y_true=np.array([1]), y_pred=np.array([g]), labels=(0, 1))
5 )
6 plt.xlabel("Confidence in correct answer")
7 plt.ylabel("Loss (lower is better)");

```



- Your loss goes to 0 as you approach 100% confidence in the correct answer.
- Your loss goes to infinity as you approach 100% confidence in the incorrect answer.
- (Optional) See also the very related [How to assign partial credit on an exam of true-false questions? \(<https://terrytao.wordpress.com/2016/06/01/how-to-assign-partial-credit-on-an-exam-of-true-false-questions/>\)](https://terrytao.wordpress.com/2016/06/01/how-to-assign-partial-credit-on-an-exam-of-true-false-questions/)

The real `LogisticRegression` is averaging this score over all training examples.

Some nice examples:

- [Scott Alexander \(<https://slatestarcodex.com/2019/01/22/2018-predictions-calibration-results/>\)](https://slatestarcodex.com/2019/01/22/2018-predictions-calibration-results/)
 - Look at how the plot starts at 50%. That is because being 40% confident of "X" is the same as being 60% confident of "not X".
- [Good Judgment Project \(<https://www.gjopen.com/>\)](https://www.gjopen.com/)

Credence Activity (time permitting: 15 min)

- Take a few minutes and assign credences or values to the claims below, in the [Google Doc](https://jamboard.google.com/d/1WbJTiNi-qt4EjvONcyO-CUxOhGxXhaRwYto28fC5Kzs/edit?usp=sharing) (<https://jamboard.google.com/d/1WbJTiNi-qt4EjvONcyO-CUxOhGxXhaRwYto28fC5Kzs/edit?usp=sharing>). Afterwards, we'll discuss.
- Do not search the answers; the point of the exercise is to evaluate how good we are at guessing.**
- Try not to be influenced by other peoples' answers! Better to pick your answers before going to the Google Doc.

- I am __ % sure that the world's longest river is between 6000km and 8000km.
- I am __ % sure that there is 4 to 8 liters of blood in an average adult human body.
- I am 99% sure that the world's tallest tree is taller than __ m.
- I am 90% sure that the world's tallest tree is taller than __ m.
- I am 50% sure that the world's tallest tree is taller than __ m.

NOTE: 100% means you are completely sure the statement is true, 0% means you are completely sure the statement is false.

Answers (if you are curious):

- Nile, 6650 km
- 5 liters
- (to 5) the world's tallest tree is [116.07 m](https://www.guinnessworldrecords.com/world-records/tallest-tree-living#:~:text=The%20tallest%20tree%20currently%20living,to%20try%20and%20protect%20it) (<https://www.guinnessworldrecords.com/world-records/tallest-tree-living#:~:text=The%20tallest%20tree%20currently%20living,to%20try%20and%20protect%20it>)

Visualizing your results

- Very powerful but at the same time can be misleading if not done properly.

Pre-viewing review from [Calling BS visualization videos](https://www.youtube.com/watch?v=T-5aLbNeGo0&list=PLPnZfvKID1Sje5jWxt-4CSZD7bUI4gSPS&index=30&t=0s) (<https://www.youtube.com/watch?v=T-5aLbNeGo0&list=PLPnZfvKID1Sje5jWxt-4CSZD7bUI4gSPS&index=30&t=0s>):

- Dataviz in the popular media.
 - e.g. [modern NYT](https://youtu.be/T-5aLbNeGo0?t=367) (<https://youtu.be/T-5aLbNeGo0?t=367>)
- Misleading axes.
 - e.g. [vaccines](https://youtu.be/9pNWVMxaFuM?t=299) (<https://youtu.be/9pNWVMxaFuM?t=299>)
- Manipulating bin sizes.
 - e.g. [tax dollars](https://youtu.be/zAg1wsYfwsM?t=196) (<https://youtu.be/zAg1wsYfwsM?t=196>)

- Dataviz ducks.
 - e.g. [drinking water](https://youtu.be/rmii1hfP6d4?t=169) (<https://youtu.be/rmii1hfP6d4?t=169>)
 - "look how clever we are about design" -> making it about me instead of about you (see last class)
- Glass slippers.
 - e.g. [internet marketing tree](https://youtu.be/59teS0SUHtI?t=285) (<https://youtu.be/59teS0SUHtI?t=285>)
- The principle of proportional ink.
 - e.g. [most read books](https://youtu.be/oNhusd3xFC4?t=147) (<https://youtu.be/oNhusd3xFC4?t=147>)

- [Demo of cleaning up a plot](https://www.darkhorseanalytics.com/blog/data-looks-better-naked) (<https://www.darkhorseanalytics.com/blog/data-looks-better-naked>)
- [Principle of proportional ink](https://serialmentor.com/dataviz/proportional-ink.html) (<https://serialmentor.com/dataviz/proportional-ink.html>) from a viz textbook.

Dataset

We'll be using [Kaggle House Prices dataset](https://www.kaggle.com/c/home-data-for-ml-course) (<https://www.kaggle.com/c/home-data-for-ml-course>), which we used in lecture 10. As usual, to run this notebook you'll need to download the data. For this dataset, train and test have already been separated. We'll be working with the train portion.

```
In [9]: 1 df = pd.read_csv("../data/housing-kaggle/train.csv")
2 train_df, test_df = train_test_split(df, test_size=0.10, random_state=1)
3 train_df.head()
```

```
Out[9]:
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	L
302	303	20	RL	118.0	13704	Pave	NaN	IR1		Lvl
767	768	50	RL	75.0	12508	Pave	NaN	IR1		Lvl
429	430	20	RL	130.0	11457	Pave	NaN	IR1		Lvl
1139	1140	30	RL	98.0	8731	Pave	NaN	IR1		Lvl
558	559	60	RL	57.0	21872	Pave	NaN	IR2		HLS

5 rows × 81 columns

```
In [10]: 1 train_df.shape
```

```
Out[10]: (1314, 81)
```

Let's separate x and y

```
In [11]:  
1 X_train = train_df.drop(columns=["SalePrice"])  
2 y_train = train_df["SalePrice"]  
3  
4 X_test = test_df.drop(columns=["SalePrice"])  
5 y_test = test_df["SalePrice"]
```

Feature types

- We have mixed feature types and a bunch of missing values.
- Now, let's identify feature types and transformations.

In [12]:

```
1 drop_features = ["Id"]
2 numeric_features = [
3     "BedroomAbvGr",
4     "KitchenAbvGr",
5     "LotFrontage",
6     "LotArea",
7     "OverallQual",
8     "OverallCond",
9     "YearBuilt",
10    "YearRemodAdd",
11    "MasVnrArea",
12    "BsmtFinSF1",
13    "BsmtFinSF2",
14    "BsmtUnfSF",
15    "TotalBsmtSF",
16    "1stFlrSF",
17    "2ndFlrSF",
18    "LowQualFinSF",
19    "GrLivArea",
20    "BsmtFullBath",
21    "BsmtHalfBath",
22    "FullBath",
23    "HalfBath",
24    "TotRmsAbvGrd",
25    "Fireplaces",
26    "GarageYrBlt",
27    "GarageCars",
28    "GarageArea",
29    "WoodDeckSF",
30    "OpenPorchSF",
31    "EnclosedPorch",
32    "3SsnPorch",
33    "ScreenPorch",
34    "PoolArea",
35    "MiscVal",
36    "YrSold",
37 ]
```

In [13]:

```

1 ordinal_features_reg = [
2     "ExterQual",
3     "ExterCond",
4     "BsmtQual",
5     "BsmtCond",
6     "HeatingQC",
7     "KitchenQual",
8     "FireplaceQu",
9     "GarageQual",
10    "GarageCond",
11    "PoolQC",
12]
13 ordering = [
14     "Po",
15     "Fa",
16     "TA",
17     "Gd",
18     "Ex",
19 ] # if N/A it will just impute something, per below
20 ordering_ordinal_reg = [ordering] * len(ordinal_features_reg)
21 ordering_ordinal_reg

```

Out[13]:

```

[[ 'Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex']]

```

In [14]:

```

1 ordinal_features_oth = [
2     "BsmtExposure",
3     "BsmtFinType1",
4     "BsmtFinType2",
5     "Functional",
6     "Fence",
7 ]
8 ordering_ordinal_oth = [
9     ["NA", "No", "Mn", "Av", "Gd"],
10    ["NA", "Unf", "LwQ", "Rec", "BLQ", "ALQ", "GLQ"],
11    ["NA", "Unf", "LwQ", "Rec", "BLQ", "ALQ", "GLQ"],
12    ["Sal", "Sev", "Maj2", "Maj1", "Mod", "Min2", "Min1", "Typ"],
13    ["NA", "MnWw", "GdWo", "MnPrv", "GdPrv"],
14 ]

```

The remaining features are categorical features.

In [15]:

```
1 categorical_features = list(
2     set(X_train.columns)
3     - set(numeric_features)
4     - set(ordinal_features_reg)
5     - set(ordinal_features_oth)
6     - set(drop_features)
7 )
8 categorical_features
```

Out[15]:

```
['SaleType',
 'MSSubClass',
 'LandContour',
 'CentralAir',
 'PavedDrive',
 'LotShape',
 'MSZoning',
 'MiscFeature',
 'Alley',
 'LotConfig',
 'Utilities',
 'MoSold',
 'MasVnrType',
 'Condition2',
 'RoofStyle',
 'RoofMatl',
 'HouseStyle',
 'Heating',
 'GarageFinish',
 'BldgType',
 'SaleCondition',
 'Foundation',
 'Condition1',
 'LandSlope',
 'Neighborhood',
 'Electrical',
 'Exterior2nd',
 'Exterior1st',
 'GarageType',
 'Street']
```

Applying feature transformations

- Since we have mixed feature types, let's use `ColumnTransformer` to apply different transformations on different features types.

In [16]:

```
1 from sklearn.compose import ColumnTransformer, make_column_transformer
2
3 numeric_transformer = make_pipeline(SimpleImputer(strategy="median"), S
4 ordinal_transformer_reg = make_pipeline(
5     SimpleImputer(strategy="most_frequent"),
6     OrdinalEncoder(categories=ordering_ordinal_reg),
7 )
8
9 ordinal_transformer_oth = make_pipeline(
10    SimpleImputer(strategy="most_frequent"),
11    OrdinalEncoder(categories=ordering_ordinal_oth),
12 )
13
14 categorical_transformer = make_pipeline(
15    SimpleImputer(strategy="constant", fill_value="missing"),
16    OneHotEncoder(handle_unknown="ignore", sparse=False),
17 )
18
19 preprocessor = make_column_transformer(
20     ("drop", drop_features),
21     (numeric_transformer, numeric_features),
22     (ordinal_transformer_reg, ordinal_features_reg),
23     (ordinal_transformer_oth, ordinal_features_oth),
24     (categorical_transformer, categorical_features),
25 )
```

Examining the preprocessed data

```
In [17]: 1 preprocessor.fit(X_train)
2 # Calling fit to examine all the transformers.
```

```
Out[17]: ColumnTransformer(transformers=[('drop', 'drop', ['Id']),
                                         ('pipeline-1',
                                          Pipeline(steps=[('simpleimputer',
                                                          SimpleImputer(strategy
= 'median')),
                                                         ('standardscaler',
                                                          StandardScaler()))])),
                                         ['BedroomAbvGr', 'KitchenAbvGr', 'LotFro
ntage',
                                          'LotArea', 'OverallQual', 'OverallCon
d',
                                          'YearBuilt', 'YearRemodAdd', 'MasVnrAre
a',
                                          'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfS
F',
                                          'TotalBsmtSF', '...
                                         ['SaleType', 'MSSubClass', 'LandContou
r',
                                          'CentralAir', 'PavedDrive', 'LotShape',
                                          'MSZoning', 'MiscFeature', 'Alley',
                                          'LotConfig', 'Utilities', 'MoSold',
                                          'MasVnrType', 'Condition2', 'RoofStyl
e',
                                          'RoofMatl', 'HouseStyle', 'Heating',
                                          'GarageFinish', 'BldgType', 'SaleCondit
ion',
                                          'Foundation', 'Condition1', 'LandSlop
e',
                                          'Neighborhood', 'Electrical', 'Exterior
2nd',
                                          'Exterior1st', 'GarageType', 'Stree
t'])])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [19]:

```

1 ohe_columns = list(
2     preprocessor.named_transformers_[ "pipeline-4" ]
3     .named_steps[ "onehotencoder" ]
4     .get_feature_names_out(categorical_features)
5 )
6 new_columns = (
7     numeric_features + ordinal_features_reg + ordinal_features_oth + oh
8 )

```

In [20]:

```

1 X_train_enc = pd.DataFrame(
2     preprocessor.transform(X_train), index=X_train.index, columns=new_c
3 )
4 X_train_enc.head()

```

Out[20]:

	BedroomAbvGr	KitchenAbvGr	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	Y
302	0.154795	-0.222647	2.312501	0.381428	0.663680	-0.512408	0.993969	
767	1.372763	-0.222647	0.260890	0.248457	-0.054669	1.285467	-1.026793	
429	0.154795	-0.222647	2.885044	0.131607	-0.054669	-0.512408	0.563314	
1139	0.154795	-0.222647	1.358264	-0.171468	-0.773017	-0.512408	-1.689338	
558	0.154795	-0.222647	-0.597924	1.289541	0.663680	-0.512408	0.828332	

5 rows × 263 columns

In [21]:

```

1 X_test_enc = pd.DataFrame(
2     preprocessor.transform(X_test), index=X_test.index, columns=new_col
3 )
4 X_test_enc.head()

```

Out[21]:

	BedroomAbvGr	KitchenAbvGr	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	Y
147	0.154795	-0.222647	-0.025381	-0.085415	0.663680	-0.512408	0.993969	
676	1.372763	4.348569	-0.454788	-0.074853	-1.491366	-3.209221	-2.351883	
1304	0.154795	-0.222647	-1.790721	-0.768279	0.663680	-0.512408	1.093350	
1372	0.154795	-0.222647	0.260890	-0.058176	0.663680	0.386530	0.894587	
1427	0.154795	-0.222647	-0.454788	0.073016	-0.773017	0.386530	-0.861157	

5 rows × 263 columns

In [22]:

```
1 X_train.shape, X_test.shape
```

Out[22]: ((1314, 80), (146, 80))

Training random forests and gradient boosted trees

```
In [23]: 1 from sklearn.ensemble import GradientBoostingRegressor
```

Let's compare sklearn's GradientBoostingRegressor to RandomForestRegressor for different values of n_estimators .

```
In [24]: 1 n_estimators_values = [3, 10, 30, 100, 300]
```

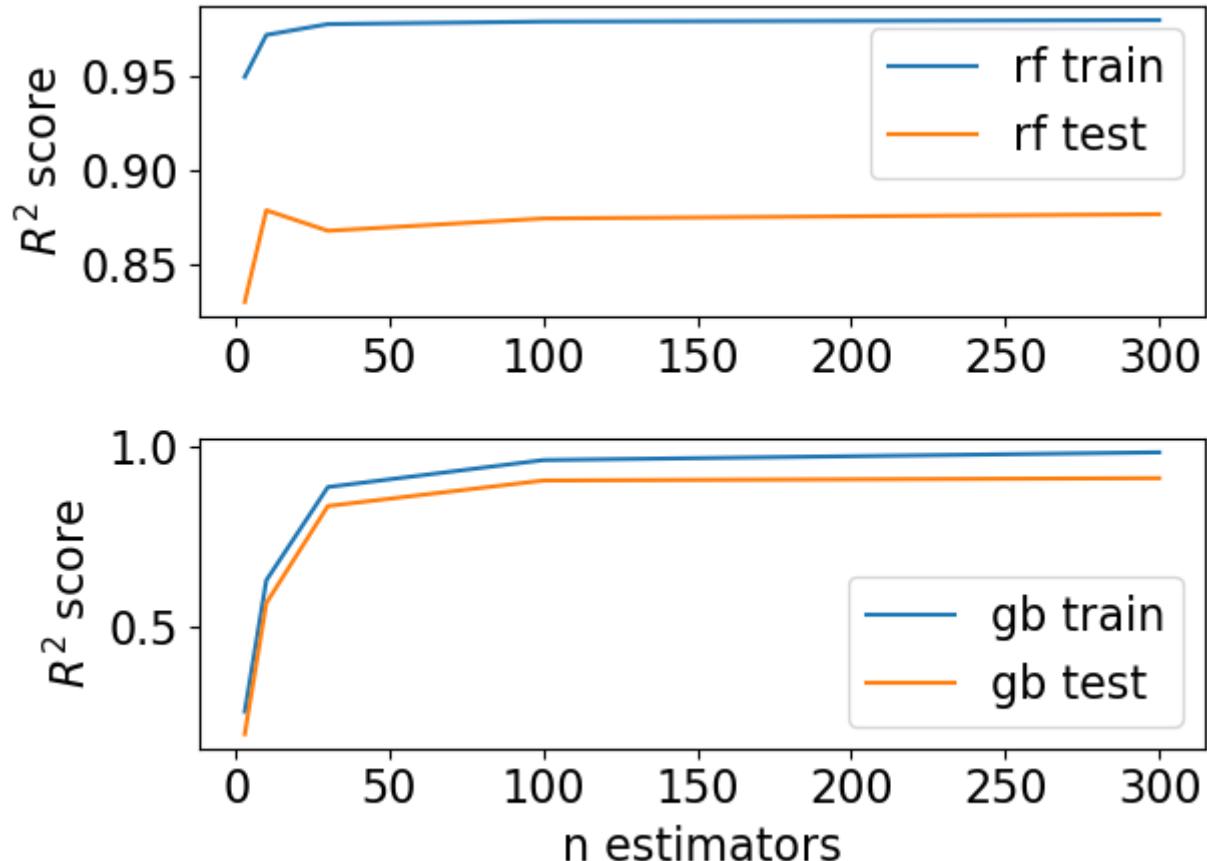
```
In [25]: 1 score_rf_train = list()
2 score_rf_test = list()
3 score_gb_train = list()
4 score_gb_test = list()
5
6 for n_estimators in n_estimators_values:
7     print(n_estimators)
8     rf = TransformedTargetRegressor(
9         RandomForestRegressor(n_estimators=n_estimators, random_state=1,
10         func=np.log1p,
11         inverse_func=np.expm1,
12     )
13     rf.fit(X_train_enc, y_train)
14     score_rf_train.append(rf.score(X_train_enc, y_train))
15     score_rf_test.append(rf.score(X_test_enc, y_test))
16
17     gb = TransformedTargetRegressor(
18         GradientBoostingRegressor(n_estimators=n_estimators, random_state=1,
19         func=np.log1p,
20         inverse_func=np.expm1,
21     )
22     gb.fit(X_train_enc, y_train)
23     score_gb_train.append(gb.score(X_train_enc, y_train))
24     score_gb_test.append(gb.score(X_test_enc, y_test))

3
10
30
100
300
```

Here is a low-quality plot that is confusing and perhaps downright misleading:

In [26]:

```
1 plt.subplot(2, 1, 1)
2 plt.plot(n_estimators_values, score_rf_train, label="rf train")
3 plt.plot(n_estimators_values, score_rf_test, label="rf test")
4 plt.ylabel("$R^2$ score")
5 plt.legend()
6 plt.subplot(2, 1, 2)
7 plt.plot(n_estimators_values, score_gb_train, label="gb train")
8 plt.plot(n_estimators_values, score_gb_test, label="gb test")
9 plt.xlabel("n estimators")
10 plt.ylabel("$R^2$ score")
11 plt.legend()
12 plt.tight_layout();
```



Let's create some visualizations.

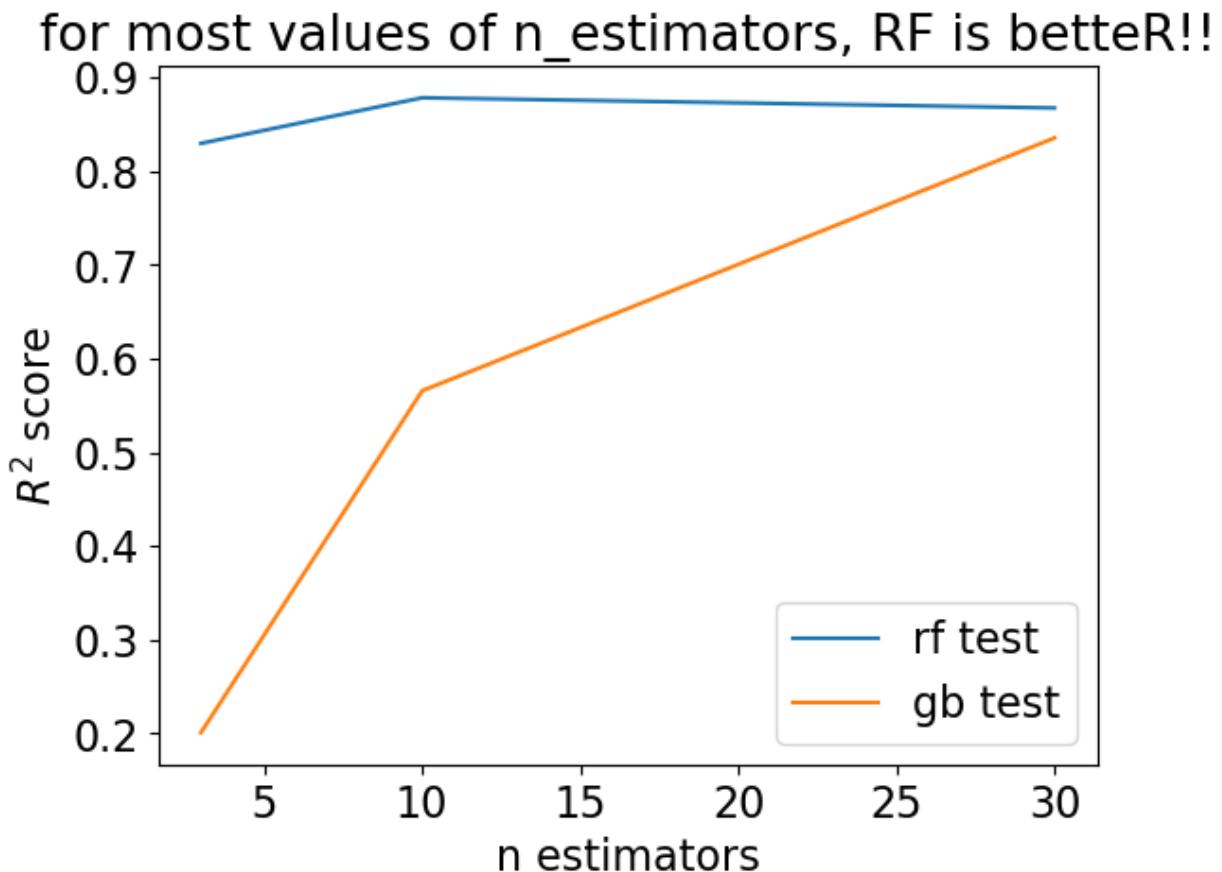
- Create a visualization that makes RF look better than GB.
- Create a visualization that makes GB look better than RF.
- Create a visualization that makes RF and GB look equally good.

Here are some misleading plots.

RF better than GB

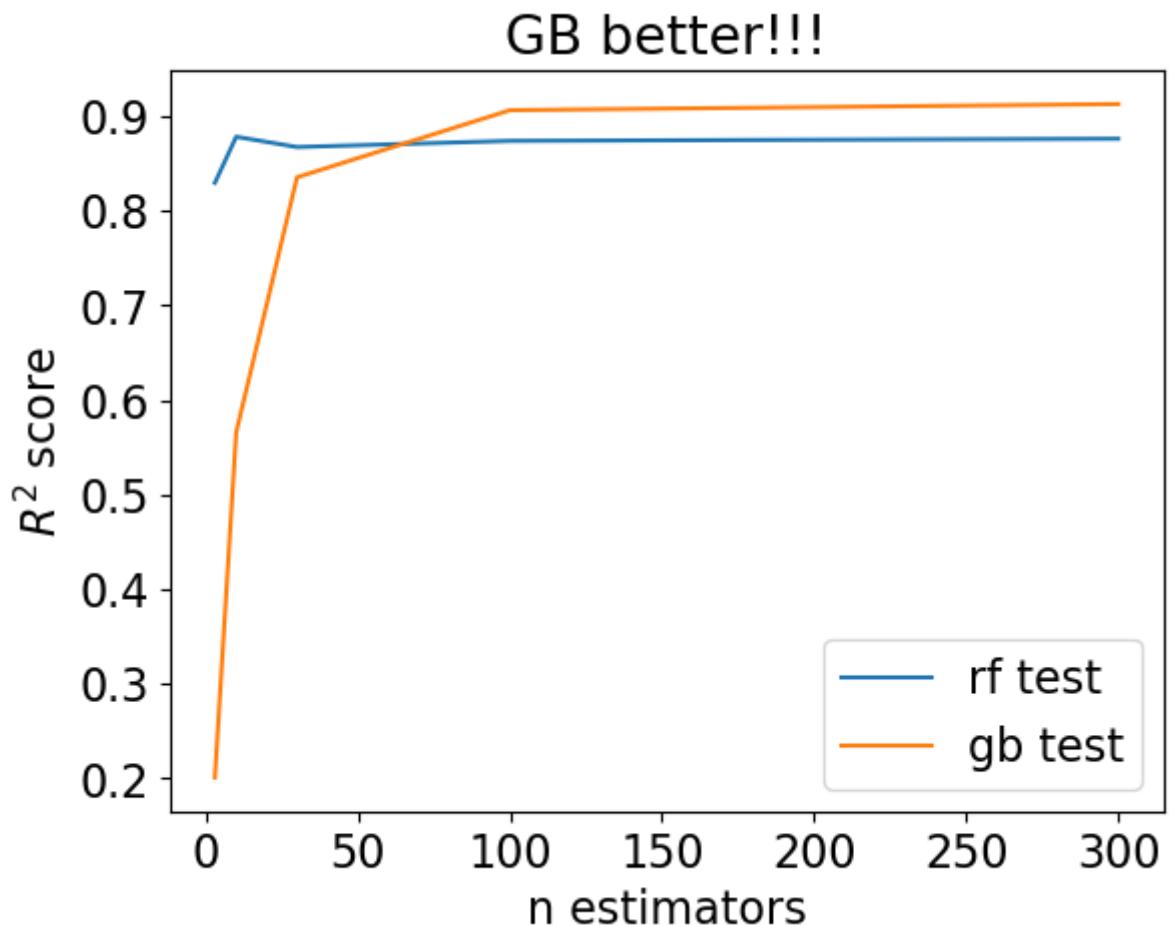
In [27]:

```
1 nmax = 3
2
3 # plt.plot(n_estimators_values[:nmax], score_rf_train[:nmax], label="rf
4 plt.plot(n_estimators_values[:nmax], score_rf_test[:nmax], label="rf te
5 # plt.plot(n_estimators_values[:nmax], score_gb_train[:nmax], label="gb
6 plt.plot(n_estimators_values[:nmax], score_gb_test[:nmax], label="gb te
7 plt.xlabel("n estimators")
8 plt.ylabel("$R^2$ score")
9 plt.legend()
10 plt.title("for most values of n_estimators, RF is betteR!!");
```



GB better than RF

```
In [28]: 1 # plt.plot(n_estimators_values, score_rf_train, label="rf train")
2 plt.plot(n_estimators_values, score_rf_test, label="rf test")
3 # plt.ylabel("$R^2$ score");
4 # plt.legend();
5 # plt.subplot(2,1,2)
6 # plt.plot(n_estimators_values, score_gb_train, label="gb train")
7 plt.plot(n_estimators_values, score_gb_test, label="gb test")
8 plt.xlabel("n estimators")
9 plt.ylabel("$R^2$ score")
10 plt.legend()
11 plt.title("GB better!!!");
```



Equally good

In [29]:

```

1 nmax = 2
2
3 # plt.plot(n_estimators_values, score_rf_train, label="rf train")
4 plt.plot(n_estimators_values[:nmax], score_rf_test[:nmax], "b", label="rf test")
5 plt.ylabel("RF $R^2$ score")
6 plt.legend(loc="lower left")
7 plt.ylim((0.8, 0.9))
8 plt.twinx()
9 # plt.plot(n_estimators_values, score_gb_train, label="gb train")
10 plt.plot(n_estimators_values[:nmax], score_gb_test[:nmax], "--r", label="gb test")
11 plt.xlabel("n estimators")
12 plt.ylabel("GB $R^2$ score")
13 plt.legend()
14 plt.ylim((-0.01, 0.70))
15 plt.title("Both equally good!!!");

```



Be critical of your visualizations and try to make them as honest as possible.

What did we learn today?

Principles of effective communication

- Concepts then labels, not the other way around.
- Bottom-up explanations.
- New ideas in small chunks.
- Reuse your running examples.
- Approaches from all angles.
- When experimenting, show the results asap.
- **It's not about you.**

- Decision variables, objectives, and context.
- How does ML fit in?
- Expressing your confidence about the results
- Misleading visualizations.

In []:

1

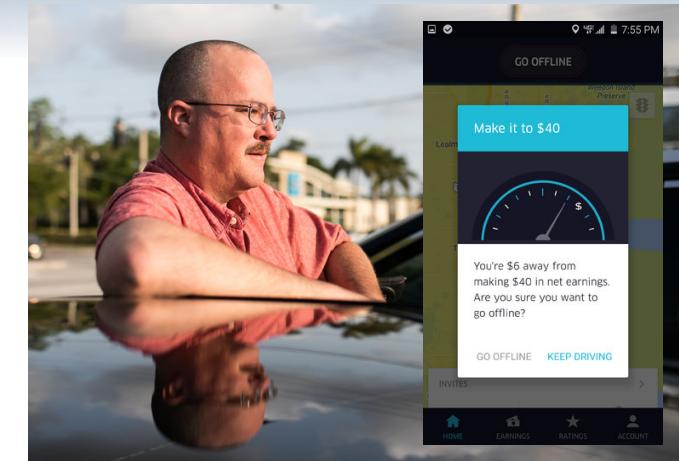


Ethics of Artificial Intelligence

Ethics of AI

Will a new technology:

- disempower **individuals vs corporations**?
⇒ user modeling; data mining; fostering addictive behaviors; developmental effects on children
- disempower **individuals vs governments**?
⇒ facilitate disinformation (deep fakes; bots masquerading as people; filter bubbles); enable qualitatively new military or security tactics
- take **autonomous actions** in a way that obscures responsibility
⇒ autonomous weapons; self-driving cars; loan approval systems
- disproportionately affect **vulnerable/marginalized groups**
⇒ automated decision making tools trained in ways that may encode existing biases



BIAS AND FAIRNESS

Human bias

Bias in people refers to our tendency to take quick decisions based on little information

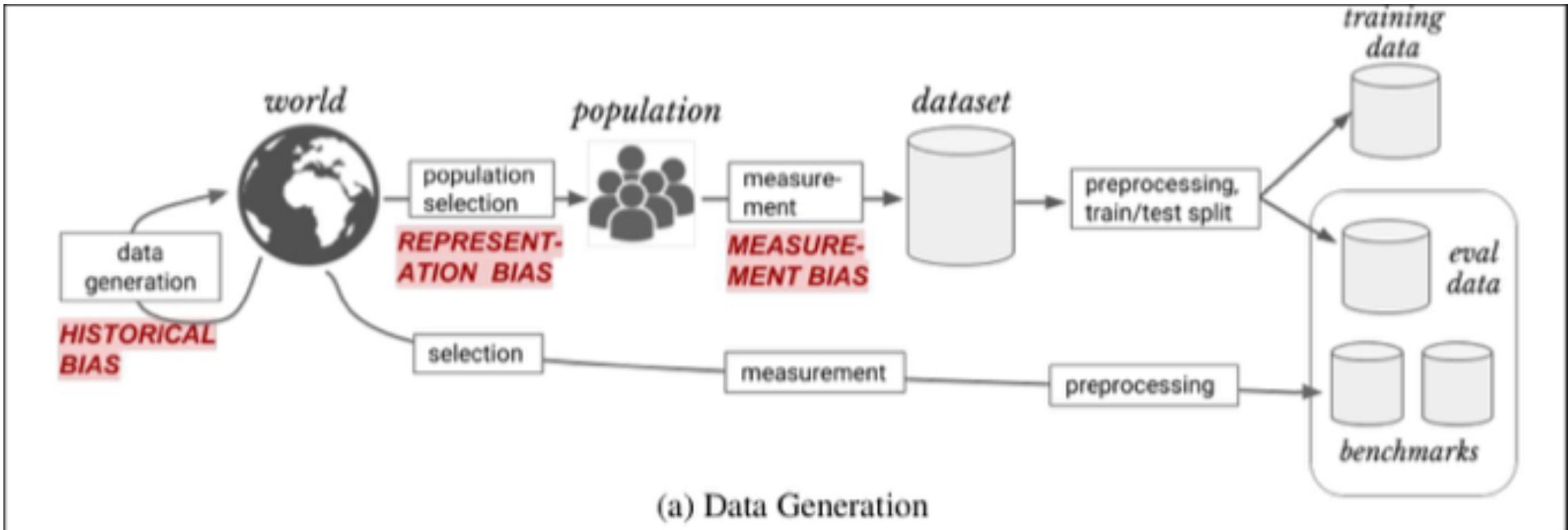
The collage consists of four separate news snippets arranged in a 2x2 grid:

- Top Left:** A snippet from Nature magazine titled "Hungry judges dispense rough justice". It includes a sub-headline "When they need a break, decision-makers gravitate towards the easy option." and author Zoë Corbyn.
- Bottom Left:** An article from the Journal of Economic Perspectives titled "Evidence on Discrimination in Mortgage Lending" by Helen F. Ladd.
- Top Right:** A snippet from The Observer titled "Racial bias in police stop and search getting worse, report reveals". It features a photograph of two police officers in high-visibility vests.
- Bottom Right:** A snippet from BBC Inside Out titled "Is it easier to get a job if you're Adam or Mohamed?". It shows two men holding signs with their names: MOHAMED ALLAM and ADAM HENTON.

Can technology have bias?



Sources of bias in ML algorithms



Historical bias

Historical bias arises when there is a misalignment between world as it is and the values or objectives to be encoded and propagated in a model. It is a normative concern with the state of the world, and exists even given perfect sampling and feature selection.

¹

Example: image search In 2018, 5% of Fortune 500 CEOs were women (Zarya, 2018). Should image search results for “CEO” reflect that number? Ultimately, a variety of stakeholders, including affected members of society, should evaluate the particular harms that this result could cause and make a judgment. This decision may be at odds with the available data even if that data is a perfect reflection of the world. Indeed, Google has recently changed their Image Search results for “CEO” to display a higher proportion of women.

Representation bias

Representation bias arises while defining and sampling a development population. It occurs when the development population under-represents, and subsequently fails to generalize well, for some part of the use population.

1

1. **The sampling methods only reach a portion of the population.** For example, datasets collected through smartphone apps can under-represent lower-income or older groups, who are less likely to own smartphones. Similarly, medical data for a particular condition may be available only for the population of patients who were considered serious enough to bring in for further screening.
2. **The population of interest has changed or is distinct from the population used during model training.** Data that is representative of Boston, for example, may not be representative if used to analyze the population of Indianapolis. Similarly, data representative of Boston 30 years ago will likely not reflect today's population.

Measurement bias

Measurement Bias arises when choosing and measuring features and labels to use; these are often proxies for the desired quantities. The chosen set of features and labels may leave out important factors or introduce group- or input-dependent noise that leads to differential performance.

3. The defined classification task is an oversimplification.

In order to build a supervised ML model, some label to predict must be chosen. Reducing a decision to a single attribute can create a biased proxy label because it only captures a particular aspect of what we really want to measure. Consider the prediction problem of deciding whether a student will be successful (e.g., in a college admissions context). Fully capturing the outcome of ‘successful student’ in terms of a single measurable attribute is impossible because of its complexity. In cases such as these, algorithm designers resort to some available label such as ‘GPA’ (Kleinberg et al. 2018), which ignores different indicators of success achieved by parts of the population.

1. **The measurement process varies across groups.** For example, if a group of factory workers is more stringently or frequently monitored, more errors will be observed in that group. This can also lead to a feedback loop wherein the group is subject to further monitoring because of the apparent higher rate of mistakes (Barocas and Selbst 2016).

2. **The quality of data varies across groups.** Structural discrimination can lead to systematically higher error rates in a certain group. For example, women are more likely to be misdiagnosed or not diagnosed for conditions where self-reported pain is a symptom (Calderone 1990). In this case, “*diagnosed with condition X*” is a biased proxy for “*has condition X*.”

Why worrying about bias in algorithms

Decisions made by a ML algorithm are:

- Cheap
- Scalable
- Automated
- Self-reinforcing
- Seemingly objective
- Often lacking appeals processes
- Not just predicting but also causing the future

Fairness in algorithms

- Nowadays, more attention is placed on algorithms being **fair**, and not just accurate.
- Fairness can be measured as:
 - demographic (or statistical) parity: population percentage should be reflected in the output classes
 - Equality of false negatives or equalized odds: constant false-negative (or both false-negative and true-negative) rates across groups.
 - Equal opportunity: equal True Positive Rate for all groups
 - Other metrics...
- Accuracy and fairness tend to be at odds with each other.
- Algorithms can be audited to test their fairness.
- *Are we ethically required to sacrifice accuracy for fairness?*

When the metric becomes the target (Goodhart's Law)

"When a measure becomes a target it ceases to be a good measure"

- Metrics introduced in the [British public healthcare system](#) (e.g. waiting time in ER) caused people to game it:
 - Cancelled scheduled operations to draft extra staff to ER
 - Required patients to wait outside the ER, e.g. in ambulances
 - Put stretchers in hallways and classified them as "beds"
 - Hospital and patients reported different wait times
- Big Data is significantly changing college applications (not in a good way)
 - Universities are given higher ranking for things such as receiving more applications, being more selective, and having more students accept their offers (while tuition is not considered)
 - This even pushed some mid-tier universities to reduce the number of offer letter sent out, especially to good students who they think would not accept. Students are losing their safety options
- Is this always undesirable? Can you think of ways to avoid this trap?

Algorithms to promote engagement

- Large, popular social media platforms use algorithms to increase user engagement
- Proposed content is designed to keep the user on the website longer
 - It also often becomes more extreme as the user follows the suggestions
 - Sometimes with very disturbing results: <https://www.npr.org/sections/thetwo-way/2017/11/27/566769570/youtube-faces-increased-criticism-that-its-unsafe-for-kids>
- They also tend to promote content that the user will agree/engage with, creating echo-chambers
 - Some theorize that echo-chambers can push people towards more extreme opinions
 - Do you agree? Should social media be required to change their recommendation algorithms to avoid these issues?
 - <https://www.pnas.org/doi/10.1073/pnas.2023301118>

Transparency and privacy trade-off

- There is a call for increased transparency in algorithms
 - What does that mean? Why does it matter?
- Transparency can sometime come at the cost of privacy and increased risks of data breaches or cyberattacks

Ethics of pricing algorithms

- Algorithms are currently used to adjust prices based on:
 - Willingness of buyer
 - Availability
- Uber surge pricing:
 - In 2014, terrorists attacked a café in Sidney, holding 10 customers and 8 employees hostage for 16 hours
 - During this time, people from the surrounding areas were evacuated. Transportation was disrupted.
 - Uber prices adapted by increasing the rate to a minimum of 100\$
 - In general, underserved (poorer) areas get worse rates under current pricing policy
 - Does Uber have an ethical obligation to correct its pricing algorithms to protect vulnerable segments of the population?

Interesting reads

